



**WYŻSZA SZKOŁA
INFORMATYKI I ZARZĄDZANIA**
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Technologie internetowe i mobilne

Hubert Dąbek

Nr albumu studenta: W65471

Internetowy serwis muzyczny

Projektowanie, implementacja i optymalizacja

Promotor: dr inż. Leszek Gajecki

PRACA DYPLOMOWA INŻYNIERSKA

Rzeszów 2025

Spis treści

Wstęp	5
1. Wprowadzenie. Opis zagadnienia	6
2. Badania literaturowe	7
2.1 Spotify	7
2.2 YouTube Music	8
2.3 SoundCloud	9
Podsumowanie.....	10
3. Koncepcja systemu	11
3.1 Zarys systemu – wprowadzenie	11
3.2 Zakres funkcjonalności	11
3.2.1 Wymagania funkcjonalne aplikacji.....	11
3.2.2 Wymagania niefunkcjonalne	12
3.3 Sekwencje funkcjonalności systemu użytkowników.	15
3.4 Diagram Klas	19
3.5 Wybór środowiska	20
3.6 Frontend.....	20
3.7 Backend	21
3.8 Serwer.....	21
3.9 Technologie deweloperskie	21
3.10 Wzorce projektowe	22
4. Charakterystyka etapów realizacji części projektowej.....	23
4.1 Etap 1: Stworzenie struktury danych.....	23
4.1.1 Utworzenie bazy danych PostgreSQL wraz z tabelami.....	23
4.2 Etap 2: Budowa backendu	24
4.2.1 Spis endpointów backendu.	24
4.3 Etap 3: Implementacja frontendu	30
4.3.1 Interfejs użytkownika.	30
4.3.2 Interfejs administratora.....	36
4.4 Etap 4: Testowanie.....	40
5. Interfejs użytkowników	42
5.1 Widoki użytkownika	42
5.1.1 Ekran logowania i rejestracji	42
5.1.2 Strona główna aplikacji	44

5.1.3	Dynamiczne wyszukiwanie utworów.....	45
5.1.4	Odtwarzacz	46
5.1.5	Dodawanie nowego utworu	46
5.2	Widoki Administratora.....	47
5.2.1	Zarządzanie użytkownikami.....	47
5.2.2	Zarządzanie wnioskami o dodanie nowych utworów.....	48
6.	Zakończenie.....	50
	Literatura	52

Wstęp

Tematem mojej pracy inżynierskiej jest analiza internetowych serwisów muzycznych oraz opracowanie w pełni funkcjonalnej i niezależnej platformy streamingowej. Inspiracją do podjęcia tego tematu był dynamiczny rozwój rynku muzycznego, coraz to większa popularność serwisów, takich jak Spotify czy YouTube Music. Projektowany przeze mnie serwis, umożliwia użytkownikom udostępnianie ulubionych utworów, tworzenie spersonalizowanych playlist oraz korzystanie z obszernej biblioteki muzycznej dostępnej w serwisie.

Analiza istniejących rozwiązań na rynku wykazała, że wiele platform boryka się z problemami w zarządzaniu treścią muzycznymi, co prowadzi do duplikacji utworów oraz opóźnień w usuwaniu nieprawidłowych wersji. Obecnie proces ten często opiera się na zgłoszeniach od użytkowników, co bywa nieefektywne i czasochłonne. Aby rozwiązać te problemy, mój serwis będzie wyposażony w nowoczesny system zgłoszeń, który umożliwia użytkownikom proponowanie nowych utworów, a administratorom ich szybką i przejrzystą weryfikację. Dzięki zaawansowanemu panelowi administracyjnemu zatwierdzanie utworów, będzie możliwe jednym kliknięciem po ich odsłuchaniu.

Projektując serwis, stawiam bardzo duży nacisk na efektywne zarządzanie treścią, intuicyjną nawigację oraz przyjazny interfejs użytkownika. Moim celem jest stworzenie platformy, która nie tylko spełni oczekiwania współczesnych miłośników muzyki, ale także wprowadzi innowacyjne rozwiązania w zarządzaniu zasobami muzycznymi.

Celem pracy jest zaprojektowanie konkurencyjnego serwisu streamingowego, który wyróżniać się będzie efektywnością zarządzania treścią, prostotą obsługi i wysoką jakością doświadczenia użytkownika, odpowiadając na potrzeby współczesnych użytkowników, szukających muzycznych wrażeń.

1. Wprowadzenie. Opis zagadnienia.

W dzisiejszym dynamicznym świecie rozwoju technologii, cyfrowa rozrywka stanowi kluczową i integralną część naszego życia społecznego. Jednym z aspektów takowej rozrywki jest dostęp do muzyki, która pełni istotną rolę w życiu i rozwoju człowieka. Muzyka wpływa na nasze emocje, decyzje i doświadczenia. Media streamingowe, są nieodłącznym serwisem wielu użytkowników komputerów, a także urządzeń mobilnych. Użytkownicy korzystają z nich nie tylko podczas wolnej chwili, ale pozwalają sobie na odsłuchiwanie utworów w trakcie pracy, treningu czy nauki.

Motywacją do wybrania tematu pracy dyplomowej, takiego jak Internetowy serwis muzyczny jest zróżnicowanie platform streamingowych i rosnące zainteresowanie muzyką w społeczeństwie. Wraz z tą tematyką nie wychodzę z ideą monopolizacji streamingu, gdyż to użytkownik decyduje, z której platformy utwór dopasowuje się do jego preferencji najlepiej. Każde z mediów reprezentuje inne zastosowanie natomiast podobne działanie wraz z efektem końcowym. Brakiem i minusem każdego z istniejących jest brak jednej z cech posiadanej przez inne medium. Profesjonalizm walczy ze swobodą, jakość dźwięku z jakością teledysku (lub jego brakiem), a mobilność i dostęp z wszechstronnym wyborem.

Wybór struktury i głównego języka programowania zarazem, jakim jest HTML skupiony był ku prostocie i dyspozycji samej aplikacji względem przeglądarek, jak i urządzeń, na których aplikacja będzie użytkowana. Funkcjonalnością strony zajmuję się JavaScript, który nadal jest jednym z najpowszechniejszych, jak i użyteczniejszych języków programowania, na którym opierane jest wiele nowych technologii. Styl został oparty o zarówno bazę, jak i technologie następcze. Językiem odpowiedzialnym za wygląd aplikacji jest CSS i Bootstrap, czyli baza gotowych kompozycji dostępnych dla każdego. Baza danych PostgreSQL. Za pobieranie danych z API popularnych serwisów odpowiada JSON – nowoczesny format wymiany danych komputerowych – nie zajmuje dużo pamięci, a zarazem jest czytelny i zrozumiały¹.

W dzisiejszym zgłoszonym społeczeństwie, media streamingowe nie są wykorzystywane jako bazy danych do słuchania ulubionych utworów, ale są one idealną platformą do eksploracji różnorodnych gatunków muzycznych, odkrywanie nowych zespołów lub artystów, jak i również dzielenia się swoimi doznaniami ze znajomymi. Temat pracy nie jest rozwiązaniem technicznym, w którym użytkownik platformy jest w stanie tylko i wyłącznie słuchać swoich ulubionych piosenek, natomiast wpisuje się w kontekst kulturowy. Aplikacja inspiruje do dzielenia się pasją do muzyki i personalnym doświadczeniem.

W dalszej części pracy dyplomowej skupimy się na istniejących rozwiązaniach w środowisku komercyjnym, których zawartość również została zaimplementowana. Rynek nie trzyma monopolu nad środowiskami, czego skutkami jest wolna wola użytkownika, z którego medium chce on korzystać. Przejdziemy również przez projektowanie, implementacje i optymalizowanie internetowego odtwarzacza muzyki, kładąc nacisk na tworzenie intuicyjnego interfejsu użytkownika. Baza danych została urozmacona o własną instancję, która przechowuje informacje o użytkowniku, a zaraz o jego preferencjach.

¹ Oracle, <https://www.oracle.com/pl/database/what-is-json/>, 24.05.2024.

2. Badania literaturowe

2.1 Spotify

Spotify jest to platforma cyfrowa do odtwarzania muzyki, jak i również podcastów. Baza utworów Spotify zapewnia do natychmiastowego dostępu do milionów treści artystów z całego świata. Jego podstawowymi funkcjami jest tworzenie playlist z oryginalnego źródła jak i podstawowe przesłuchiwanie wspomnianych utworów².

Funkcjonalność Spotify:

- łatwo dostępne utwory muzyczne i podcasty
- możliwość modelowania i łączenia playlist – podcasty i utwory jesteśmy w stanie łączyć w obszarze utworzonej przez użytkownika playlisty.
- obserwowanie ulubionych artystów
- publiczne i prywatne playlisty
- treści polecające – zbiory piosenek wygenerowane przez platformy, bazujące na słuchanych: artystach, gatunkach, albumach.
- inteligentne rekomendacje – opcja losowego odtwarzania zbioru muzyki, która rozszerza ugrupowanie o kolejne utwory o podobnym charakterze
- informacje o najbliższych koncertach artysty.

Elementy wspólne:

- odsłuchiwanie muzyki w formacie MP3
- tworzenie playlist – rekomendacje względem upodobań użytkownika
- publiczne playlisty innych osób korzystających z serwisu
- wersja internetowa aplikacji
- pobieranie utworów

Elementy różne:

- aplikacja mobilna i desktopowa
- korzystanie z algorytmów sztucznej inteligencji w celu dobrania jak najlepszych rekomendacji dla klienta
- podcasty – Spotify jako platforma streamingowa posiada również w swojej ofercie miejsce do udostępniania/streamowania podcastów
- jedynie oryginalne utwory autoryzowanych artystów.

Zalety:

²Spotify Support, <https://support.spotify.com/pl/article/what-is-spotify/>, 26.05.2024.

- intuicyjny interfejs aplikacji – łatwy w nawigacji zarówno w aplikacji mobilnej, jak i stacjonarnej
- opcja karaoke – tekst postępujący równo z treścią piosenki
- integracja z innymi aplikacjami takimi jak głośniki, telewizory, samochody itd.
- wysoka jakość odtwarzanych utworów

Wady:

- brak dodawania własnych utworów – w Spotify, aby dodać własny utwór, administracja musi najpierw zweryfikować użytkownika w celu zmiany jego profilu z użytkownika na artystę
- płatna subskrypcja – koszt subskrypcji premium – wysoki w porównaniu do innych tego typu aplikacji
- reklamy w wersji darmowej – uciążliwe dla użytkowników darmowej wersji Spotify
- zużycie danych – wysoka jakość streamowanej muzyki zużywa dużą ilość danych mobilnych

2.2 YouTube Music

YouTube Music jest to wyciągnięcie ręki marki Google do swojej najliczniejszej grupy odbiorców, czyli słuchaczy muzyki. Aplikacja ta posiada jeden jedyny powód dlaczego została stworzona, a więc słuchanie muzyki, jak również treści ściśle związanej z muzyką. Platforma ta korzysta z ogromnej bazy utworów YouTube. Wszystkie treści są identyfikowane przez partnerów YouTube, a więc nie wszystko co chcemy umieścić na platformie będzie widoczne, ponieważ treści są filtrowane. YouTube Music został stworzony dla miłośników muzyki, całych rodzin, a także graczy, którzy to w trakcie swojej pracy lub ulubionej rozrywki przeżywają właśnie z doświadczeniem swojej ulubionej muzyki³.

Funkcjonalność YouTube Music:

- aplikacje dostępne i skompatybilizowane do wszystkich platform.
- synchronizacja z Kontem Google – przechodzenie pomiędzy aplikacjami marki Google jak i również dostęp do spersonalizowanych rekomendacji.
- teksty piosenek – tekst piosenek wyświetlany równo podczas odtwarzania utworu

Elementy wspólne:

- remiksy i fanowskie wersje / Covery
- możliwość oceny utworu
- pobieranie utworów
- odsłuchiwanie piosenek w wysokiej jakości dźwięku

Elementy różne:

- integracja z Google Assistant – jako produkt Google, YouTube Music jest kompatybilne z polecaniami Google Assistant

³ Google Support, <https://support.google.com/youtubemusic/answer/6312991?hl=pl>, 28.05.2024

- aplikacja mobilna i desktopowa

- teledyski

Zalety:

- większość funkcji dostępna w darmowej wersji z reklamami

- możliwość wyboru jakości dźwięku

- jedna subskrypcja zarówno dla YouTube Music, jak i zwykłego YouTube.

Wady:

- płatna subskrypcja odblokowująca funkcje takie jak np. pobieranie

- brak niektórych artystów

- koszt subskrypcji

2.3 SoundCloud

Soundcloud jest najbardziej niezwykłym serwisem w tym zestawieniu, gdyż sam portal nie służy tylko do słuchania muzyki lub odkrywania nowych artystów. Jest on również medium, które pozwala artyście na stały kontakt ze swoją wspólnotą poprzez komentowanie i ocenę utworów. Platforma jest na to bardzo otwarta i daje miejsce artyście na udostępnianie wpisów dotyczących swojej twórczości. Soundcloud jest skierowany ku mniejszym artystom, którzy dopiero zaczynają swoją karierę muzyczną. Medium to zostało rozpromowane, jako wejście do branży muzyki, przez co dużo wielkich wytwórni szuka i czyha tam na nowe gwiazdy światowej muzyki⁴.

Funkcjonalność:

- komentowanie treści w czasie rzeczywistym

- łatwość publikacji

- zaawansowane narzędzia analityczne

Elementy wspólne:

- prostota w udostępnianiu swoich utworów

- możliwość polubienia utworu i sprawdzenia ilości polubień

- odtwarzanie muzyki w wysokiej jakości dźwięku

Elementy różne:

- tworzenie wspólnoty w serwisie

- możliwość komentowania treści w czasie rzeczywistym utworu

- aplikacja mobilna App Store i Google Play

⁴ eMastered, <https://emastered.com/pl/blog/royalty-free-music-soundcloud>, 09.02.2025

- etapy płatnej subskrypcji

Zalety:

- ograniczona wersja darmowej licencji – SoundCloud pomimo etapów subskrypcji pozwala użytkownikowi wersji darmowej na takie funkcje, jak słuchanie online – jest ono ograniczone, ale jako jedyna aplikacja daje taki dostęp

- w Soundcloud znajdziemy głównie niszowych artystów.

- unikalność – przez ilość treści każdy znajdzie coś dla siebie na platformie.

Wady:

- brak popularnych utworów

- mało intuicyjny interfejs użytkownika

- ograniczone treści mainstreamowe – popularni artyści rzadko decydują się na udostępnianie swoich utworów na platformie

Podsumowanie

Każda z analizowanych platform – Spotify, YouTube Music, SoundCloud – ma swoje unikalne cechy, zalety i wady. Spotify wyróżnia bogaty katalog muzyczny z inteligentnymi rekommendacjami, ale wymaga płatnej subskrypcji do pełnoprawnej funkcjonalności aplikacji. YouTube Music przyciąga użytkowników swoją integracją z ekosystemem Google oraz dostępem do telewizorów, ale również posiada płatne funkcje, które rozszerzają bazową dostępność. SoundCloud wyróżnia się możliwością łatwego udostępniania utworów przez mniejszych artystów i interakcją wykonawcy z fanami, lecz umniejsza aplikacji brak popularnych utworów. Wybór odpowiedniej platformy zależy od indywidualnych potrzeb i preferencji użytkownika. Każda z nich przyczynia się do różnorodności dostępnych opcji w świecie muzyki.

3. Koncepcja systemu

3.1 Zarys systemu – wprowadzenie

W realizowanym systemie bierzemy pod uwagę trzy typy użytkowników: użytkownik niezalogowany, użytkownik zalogowany oraz również administrator systemu. Do ogólnych zadań aplikacji będą należeć:

- Słuchanie muzyki
- Przeglądanie utworów i trendów
- Tworzenie playlist

Aplikacja ma na celu dostęp do szerokiego katalogu muzyki, który użytkownicy mogą tworzyć sami. Podziały i możliwości każdego z typów użytkownika będzie dziedziczył funkcje kolejnego z klasy poprzedzającej, przez co hierarchia układa się w sposób następujący: użytkownik niezalogowany nie może korzystać z aplikacji, a jedną funkcjonalnością dostępną dla niego jest logowanie i rejestracja, natomiast administrator posiada wszystkie uprawnienia łącznie z edycją innych użytkowników.

Użytkownik zalogowany dostaje dostęp do przeszukiwania bazy utworów, odsłuchiwać wybrane, a także tworzyć z nich playlist aby ulubiona składanka została zapisana w systemie i można by odsłuchać ją następnym razem. Oprócz tworzenia playlist, użytkownik ma prawo do edytowania przez siebie stworzonej składanki.

3.2 Zakres funkcjonalności

3.2.1 Wymagania funkcjonalne aplikacji

Podstawowe funkcje aplikacji:

- Rejestracja i logowanie użytkowników: System umożliwia tworzenie kont użytkowników oraz logowanie się do aplikacji.
- Personalizacja: Użytkownicy mogą tworzyć i zarządzać własnymi playlistami oraz oznaczać ulubione utwory.
- Wyszukiwanie muzyki: Możliwość wyszukiwania utworów, artystów i albumów zintegrowanych z różnych platform streamingowych.
- Odtwarzanie muzyki: Odtwarzanie utworów w przeglądarce z opcją sterowania (odtwarzanie, pauza, przewijanie).
- Rekomendacje: System rekomendacji muzycznych na podstawie preferencji i historii odtwarzania użytkownika.
- Udostępnianie: Opcja dzielenia się playlistami i utworami ze znajomymi.

Integracja z zewnętrznymi serwisami

- API Integracja: Aplikacja integruje się z API popularnych serwisów streamingowych (Spotify, YouTube Music, SoundCloud).
- Pobieranie danych z API: Użycie JSON do pobierania i przetwarzania danych zewnętrznych API.

Zarządzanie Bazą Danych

- Baza danych użytkowników: Przechowywanie danych użytkowników, w tym informacji o ich preferencjach muzycznych.
- Baza danych muzyki: Przechowywanie informacji o utworach, albumach i artystach.
- Historia odtwarzania: Przechowywanie historii odtwarzania utworów przez użytkowników.

Tab. 1. Wymagania funkcjonalne

Nazwa	Opis	Priorytet
W01	Rejestracja użytkownika	Wysoki
W02	Logowanie użytkownika	Wysoki
W03	Przeglądanie muzyki	Średni
W04	Wyszukiwanie muzyki	Wysoki
W05	Odtwarzanie utworu	Wysoki
W06	Tworzenie playlist	Średni
W07	Zarządzanie playlistami	Średni
W08	Oznaczanie utworów jako ulubione	Średni
W09	Udostępnianie playlist	Niski
W10	Rekomendacje muzyczne	Średni
W11	Historia odtwarzania	Niski
W12	Zarządzanie kontem użytkownika	Średni
W13	Administracja użytkowników i treści	Wysoki
W14	Integracja z systemami streamingowymi	Wysoki

Źródło: opracowanie własne

3.2.2 Wymagania niefunkcjonalne

Wydajność

- Czas odpowiedzi: Aplikacja powinna zapewniać szybki czas odpowiedzi na akcje użytkownika, maksymalnie do 2 sekund.
- Skalowalność: System musi być skalowalny, aby obsługiwać rosnącą liczbę użytkowników i odtwarzanych utworów.

Użyteczność

- Intuicyjny interfejs: Aplikacja musi posiadać łatwy w obsłudze i intuicyjny interfejs użytkownika.
- Responsywność: Strona musi być responsywna, dostosowując się do różnych urządzeń (komputery, tablety, smartfony).

Bezpieczeństwo

- Ochrona danych: Zapewnienie bezpieczeństwa danych użytkowników, w tym szyfrowanie haseł i zabezpieczenie transmisji danych.
- Autoryzacja i uwierzytelnianie: Implementacja mechanizmów autoryzacji i uwierzytelniania użytkowników.

Tab. 2. Wymagania niefunkcjonalne

Cecha	Opis	Miara
Wydajność	Szybkość działania	Czas odpowiedzi poniżej 2s
Skalowalność	Obsługa dużej liczby użytkowników	Obsługa 10000+ aktywnych użytkowników
Bezpieczeństwo	Ochrona danych	Szyfrowane dane użytkowników
Dostępność	Czas pracy systemu	99,9% SLA
Użyteczność	Intuicyjność systemu	Dobra ocena użytkowników w trakcie testów
Kompatybilność	Działanie na różnych platformach	Windows i Android
Obsługa błędów	Radzenie sobie z błędami	Odpowiedni komunikat
Wydajność serwera	Obciążenie serwera	Nie więcej niż 70% przy 10000 użytkowników
Reaktywność	Szybkość działania strony	Odświeżanie maksymalnie 3 sekundy

Źródło: opracowanie własne

Każdy z rodzaju użytkowników posiada różne funkcjonalności. Użytkownicy są tak zaprojektowani, że każdy jeden dziedziczy funkcjonalności po poprzednim. Ich dokładne uprawnienia zostały zaprezentowane w następujących tabelach.

Tab. 3. Funkcjonalności użytkownika niezalogowanego

Aktor	Użytkownik niezalogowany
Przypadki użycia	Rejestracja użytkownika
	Logowanie użytkownika

Źródło: Opracowanie własne

Tab. 4. Funkcjonalności użytkownika zalogowanego

Aktor	Użytkownik zalogowany
Przypadki użycia	Wylogowanie
	Przeglądanie katalogu muzyki
	Wyszukiwanie muzyki
	Odtworzenie utworu
	Tworzenie playlist
	Zarządzanie playlistami
	Oznaczanie utworu jak polubionego
	Udostępnienie playlist
	Otrzymywanie rekommendacji
	Zarządzanie kontem użytkownika

Źródło: Opracowanie własne

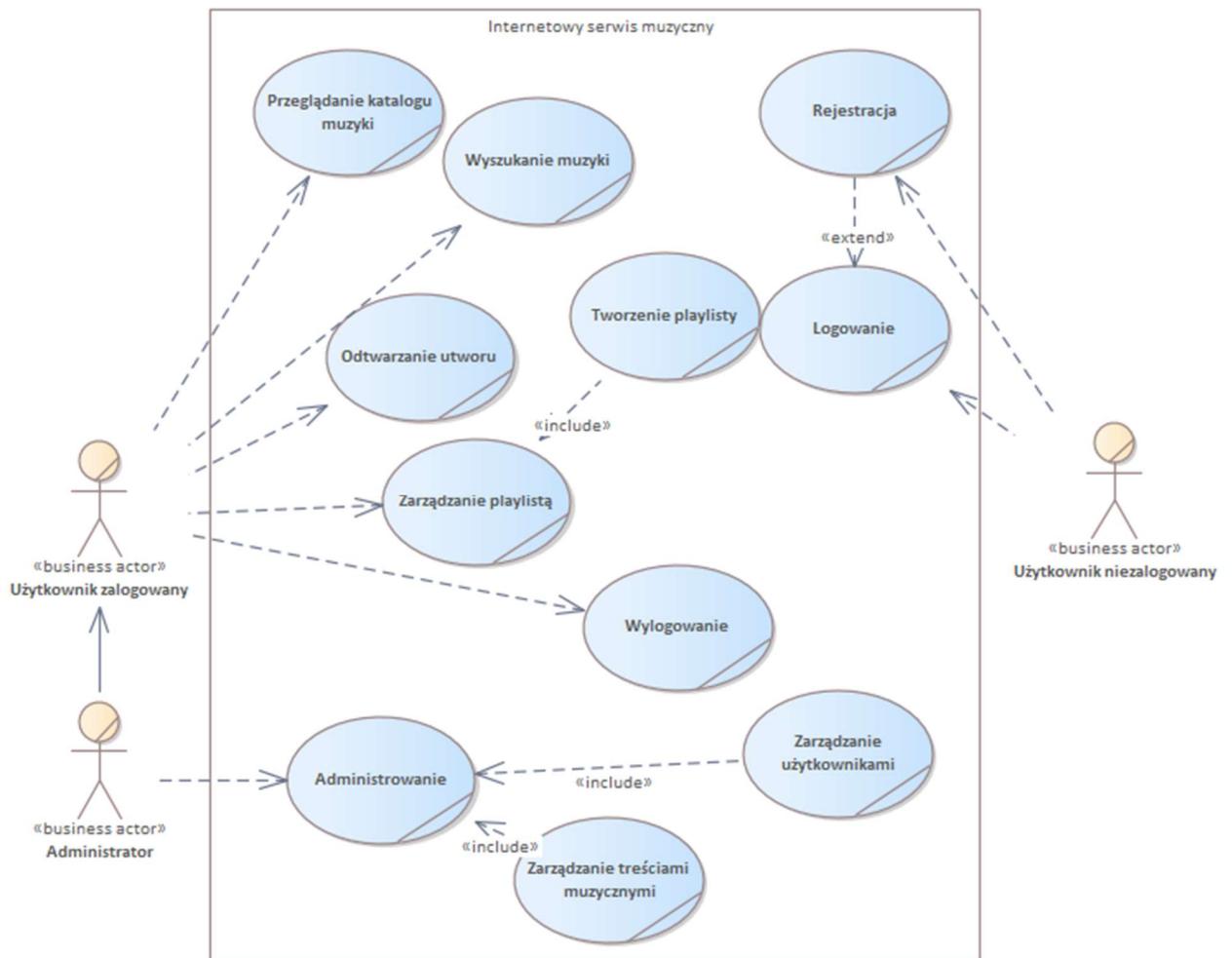
Tab. 4. Funkcjonalności Administrator

Aktor	Użytkownik zalogowany
Przypadki użycia	Wylogowanie
	Przeglądanie katalogu muzyki
	Wyszukiwanie muzyki
	Odtworzenie utworu
	Tworzenie playlist
	Zarządzanie playlistami
	Oznaczanie utworu jak polubionego
	Udostępnienie playlist
	Otrzymywanie rekomendacji
	Zarządzanie kontem użytkownika
	Zarządzanie użytkownikami
	Zarządzanie treściami

Źródło: Opracowanie własne

Według diagramu przypadków użycia i tabel umieszczonych powyżej, można zauważyć, że użytkownicy aplikacjami nie są do końca osobnymi bytami. Użytkownik istnieje i próbuje się zalogować do aplikacji, jeśli nie posiada konta użytkownika to dokonuje rejestracji, a następnie logowania i staje się użytkownikiem zalogowanym. Użytkownik zalogowany dostaje już wszystkie funkcjonalności aplikacji oprócz funkcjonalności administracyjnych, które przypadają tylko i wyłącznie administratorowi systemu.

Rys. 1. Diagram przypadków użycia użytkowników



Źródło: Opracowanie własne

3.3 Sekwencje funkcjonalności systemu użytkowników.

W niniejszym podrozdziale opiszę sekwencję kilku wybranych funkcjonalności systemu związanych z interakcją użytkownika z aplikacją. Omówione zostaną etapy logowania, przeglądanie katalogu muzyki oraz odtwarzanie utworu muzycznego w aplikacji. Każda z tych operacji wymaga odpowiedniej komunikacji między użytkownikiem, przeglądarką, serwerem, bazą danych oraz z serwisem AWS S3 w przypadku odsłuchiwanego muzyki. Przedstawione zostaną schematy opisujące przepływ danych oraz procesy zachodzące w systemie na każdym etapie interakcji. Dzięki temu podrozdziałowi będzie możliwe zrozumienie działanie aplikacji oraz mechanizmów zapewniających płynność i bezpieczeństwo operacji wykonywanych przez użytkowników.

Logowanie użytkownika

Użytkownik -> Przeglądarka: Wprowadzenie danych logowania.

Przeglądarka -> Serwer: Wysłanie danych logowania (żądanie HTTP POST).

Serwer -> Baza danych: Sprawdzenie danych logowania.

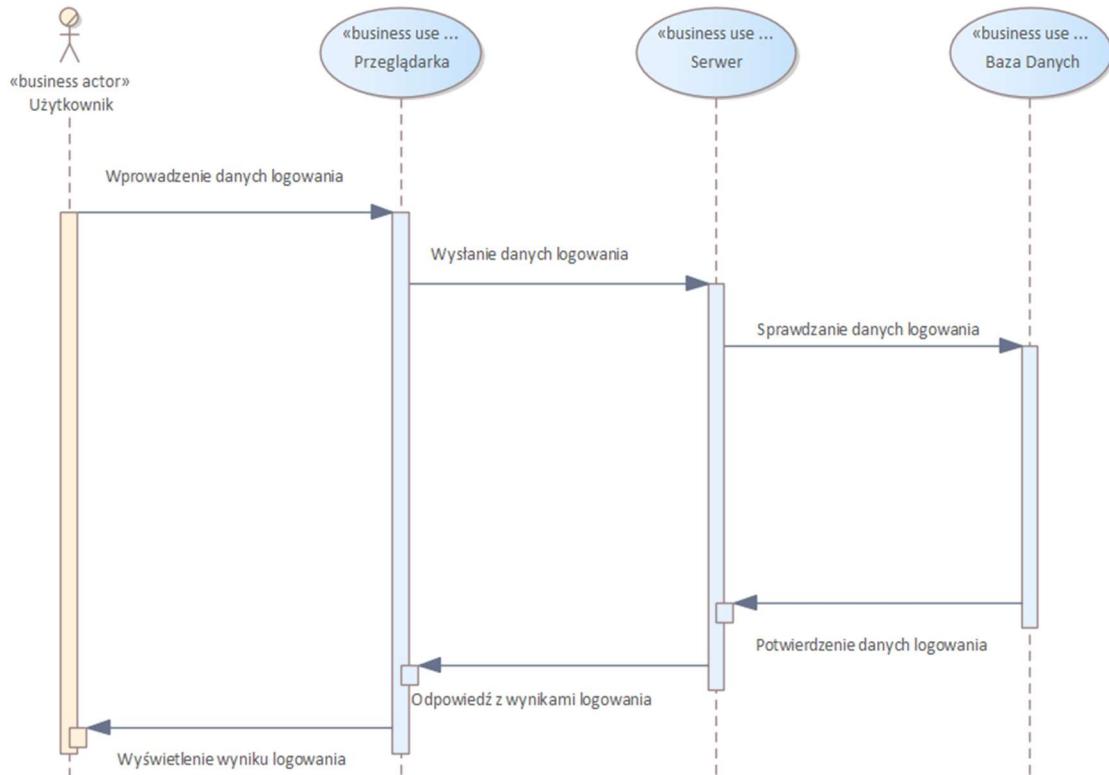
Baza danych -> Serwer: Potwierdzenie danych logowania (poprawne/niepoprawne).

Serwer -> Przeglądarka: Odpowiedź z wynikami logowania (żądanie HTTP POST).

Przeglądarka -> Użytkownik: Wyświetlenie wyniku logowania (sukces/błąd).

Diagram sekwencji przedstawia proces logowania użytkownika w aplikacji, opisując interakcje pomiędzy: użytkownikiem, przeglądarką, serwerem i bazą danych. Użytkownik na samym początku wprowadza dane logowania do przeglądarki. Następnie przeglądarka przesyła do serwera żądanie POST w celu sprawdzenia danych logowania użytkownika. Serwer przy pomocy endpointu odwołuje się do danych logowania w bazie danych, w której to wszystkie informacje są umieszczone. Baza danych potwierdza dane użytkownika, a następnie serwer tworzy JsonWebToken, który tworzy sesje użytkownika w przeglądarce. Użytkownik może korzystać z możliwości swojej roli.

Rys. 2. Diagram sekwencji logowania użytkownika niezalogowanego



Źródło: Opracowanie własne

Przeglądanie katalogu muzyki

Użytkownik -> Przeglądarka: Żądanie przeglądania katalogu muzyki.

Przeglądarka -> Serwer: Żądanie danych katalogu muzyki (żądanie HTTP GET).

Serwer -> Baza danych: Żądanie pobrania danych muzycznych.

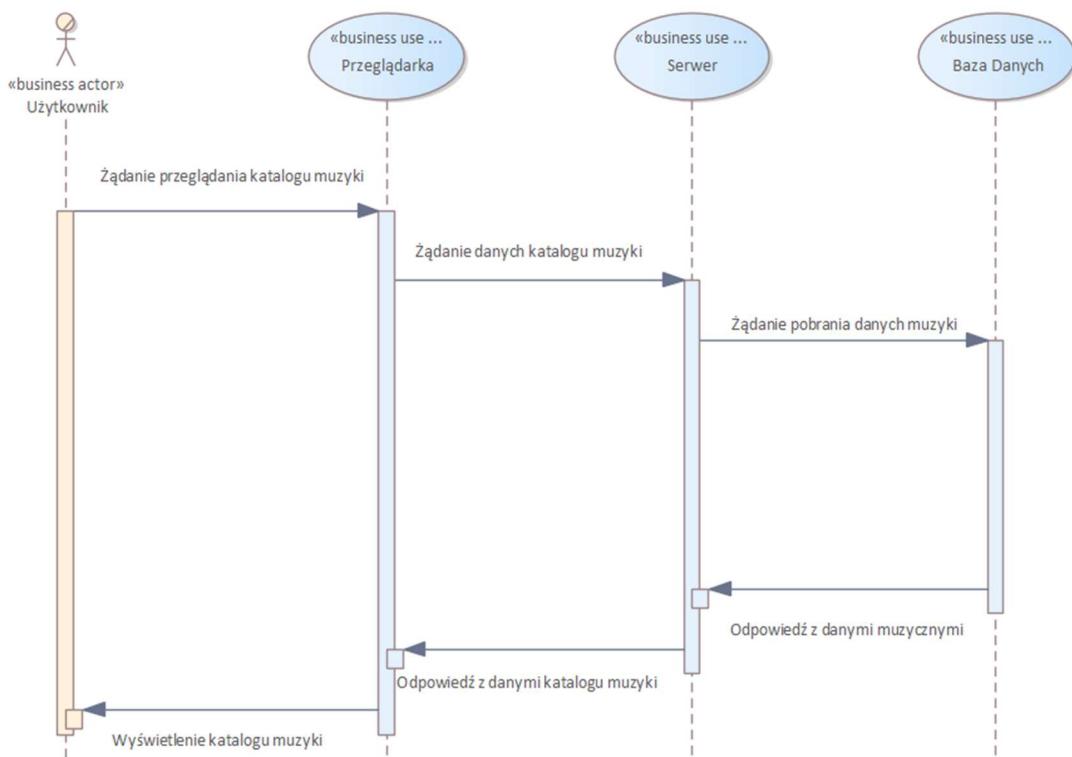
Baza danych -> Serwer: Odpowiedź z danymi muzycznymi.

Serwer -> Przeglądarka: Odpowiedź z danymi katalogu muzyki (żądanie HTTP GET).

Przeglądarka -> Użytkownik: Wyświetlenie katalogu muzyki.

Diagram sekwencji przedstawia proces przeglądania katalogów muzycznych pomiędzy użytkownikiem, przeglądarką, serwerem i bazą danych. W pierwszej kolejności użytkownik przez próbę otwarcia katalogu muzycznego zapytuje przeglądarki. W następnej części przeglądarka żąda danych od serwera, a serwer dorytuje o nie bazę danych. Baza danych zwraca wszystkie informacje przez serwer, aż trafią do przeglądarki, w której są one uporządkowane dla użytkownika.

Rys. 3. Diagram sekwencji przeglądanie katalogu muzyki



Źródło: Opracowanie własne

Odtwarzanie utworu

Użytkownik -> Przeglądarka: Żądanie odtworzenia utworu.

Przeglądarka -> Serwer: Żądanie odtworzenia utworu (żądanie HTTP GET).

Serwer -> Baza danych: Żądanie URL.

Baza danych -> AWS S3Bucket: Żądanie pobrania utworu

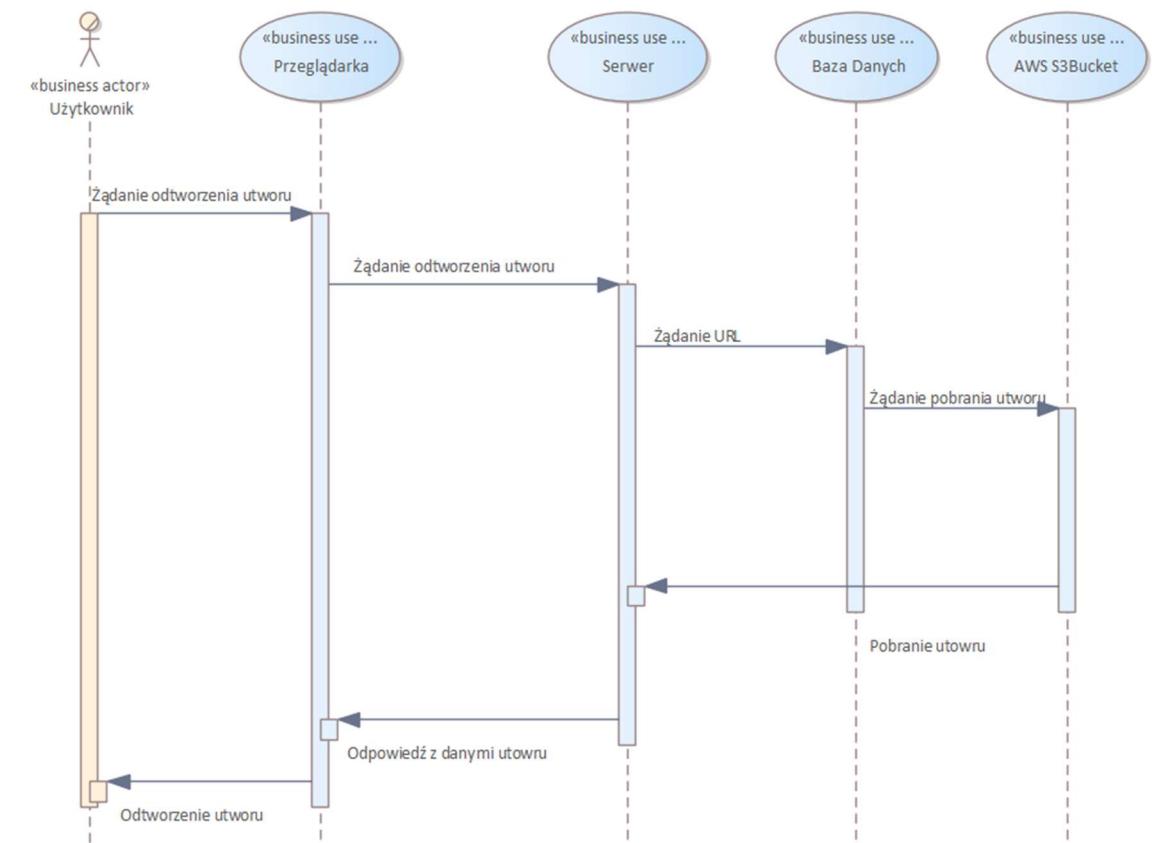
AWS S3Bucket -> Serwer: Pobranie utworu.

Serwer -> Przeglądarka: Odpowiedź z danymi utworu (żądanie HTTP GET).

Przeglądarka -> Użytkownik: Odtwarzanie utworu.

Diagram sekwencji ukazuje proces i drogę użytkownika do odtwarzania utworu. Proces ten przedstawia użytkownika, jego zapytanie do przeglądarki, następnie za pomocą żądania GET pobranie informacji o utworze serwera. Serwer dorytuje o wszystko bazę danych. Baza danych posiada URL piosenki. W kolumnie URL w bazie danych jest ukryte ID, które razem z połączonym kluczem daje dostęp do pobrania utworu z serwisu AWS S3Bucket. Plik jest pobierany na serwer, gdzie odbywa się streamowanie już po stronie przeglądarki.

Rys. 4. Diagram sekwencji odtwarzania utworu

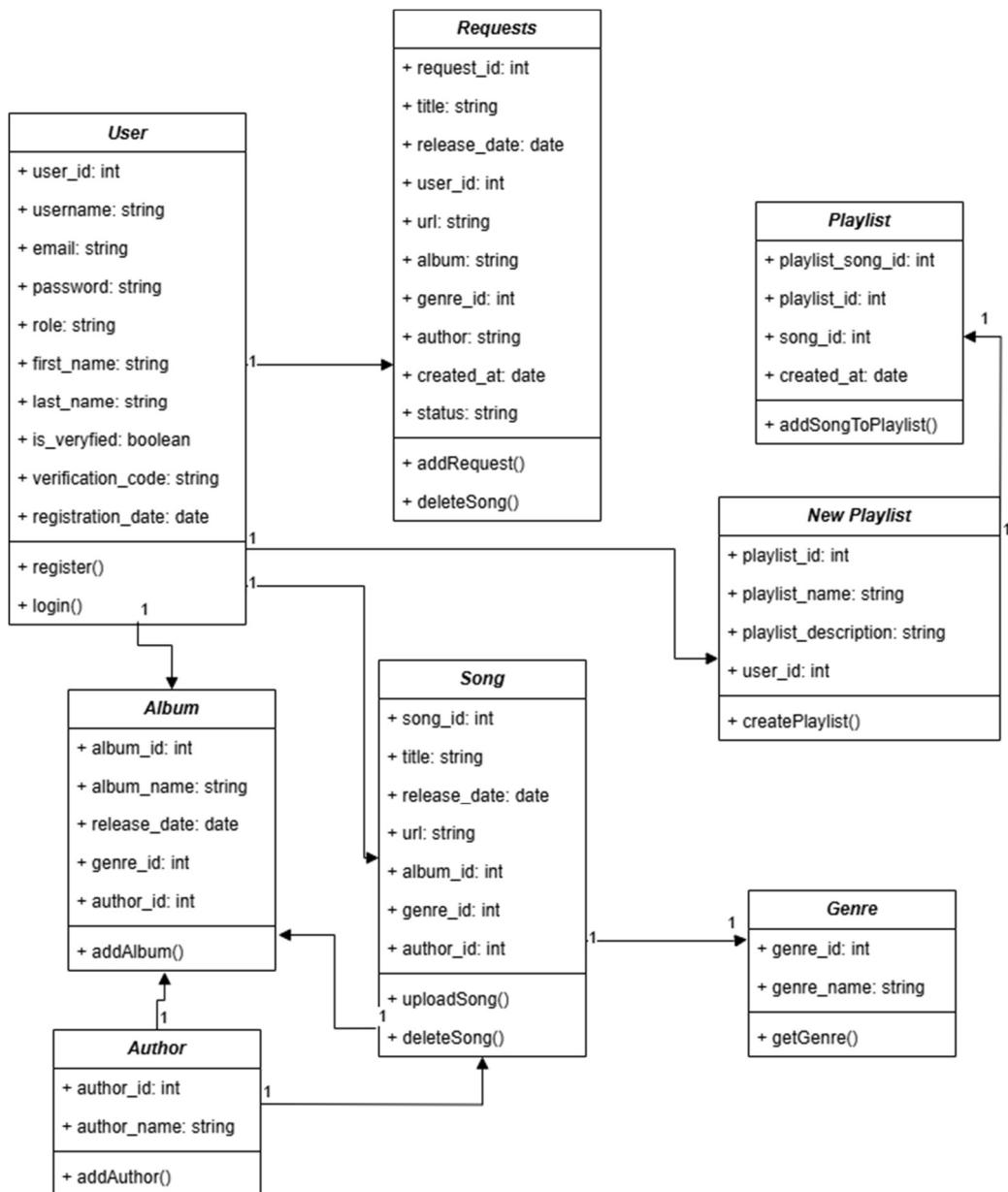


Źródło: Opracowanie własne

3.4 Diagram Klas

Diagram przedstawia główne elementy systemu, ich atrybuty i niektóre metody oraz relacje między nimi. Aplikacja pozwala na odsłuchiwanie muzyki, a także dodawanie ich na samym początku w postaci wniosku, a później jako utwór w aplikacji, którego użytkownik może umieścić w składance muzycznej i odsłuchiwać dowoli. System można podzielić na klasy, w których wyróżnimy użytkowników systemu, zarządzanie treścią muzyczną oraz obsługę techniczną. Użytkownicy będą mogli utworzyć swoje konta, a po niezbędnej autoryzacji korzystać z systemu. W obsłudze treścią wymienimy takie czynności jak tworzenie nowych składanek, odsłuchiwanie utworów ulubionych wykonawców i dodawanie ich do wcześniej stworzonych playlist. Mamy również zaplecze techniczne, czyli w wypadku klas wnioski o dodani utworów. Sprawdzanie plików, a w szczególności ich rozszerzeń jest niezbędne do funkcjonowania owej aplikacji. Diagram klas przedstawia częściową strukturę aplikacji, podkreślając rolę użytkowników, treści muzycznej i zaplecza technicznego.

Rys. 5. Diagram klas



Źródło: Opracowanie własne

3.5 Wybór środowiska

Platformowa docelowa aplikacji jest jednostka komputerowa i przeglądarka internetowa. W dzisiejszych czasach dostęp do przeglądarki ma większość urządzeń, a więc i komputery stacjonarne, ale również i mobilne wspierające przeglądarki w szerokim świetle będą w stanie uruchomić i używać aplikacji. Oprócz wyboru środowiska, w którym gotowy produkt będzie działał zamierzam opisać pokrótkę każdy element użyty w trakcie tworzenia aplikacji i również dalszego jej rozwijania.

3.6 Frontend

W sieci WWW pierwsze co pojawi się przed naszymi oczyma, zaraz po wpisaniu adresu domeny jest strona internetowa, a w bardziej szczegółowym języku właśnie Frontend. Frontend jest odpowiedzialny za wszystko, co rzuca nam się w oczy, czyli niżej opisana: struktura (HTML), wygląd (CSS) i funkcjonalność (JavaScript).

HTML – (HyperText Markup Language) – w najprostszych słowach jest technologią służącą do tworzenia stron internetowych. Jego charakterystyczną cechą zawartą pod słowem rozwinięcia skrótu „Markup” czyli znacznik, którego określa się jako tag. Język ten jest specyficzny, gdyż nie posiada on logiki typowej dla programowania jak inne odpowiedniki, takie jak JavaScript czy Python. HTML odpowiada za strukturę strony budowaną ze wcześniej wspomnianych znaczników. Jego głównym zadaniem jest wskazywanie układu strony i tego, gdzie każdy z wymaganych elementów ma się znajdować. Na tle innych języków potrzebnych do realizacji strony internetowej, HTML służy do deklaracji elementów wewnętrz strony. Strukturą języka jest wcześniej wspomniany znacznik, który otwiera i zamyka element strony, wewnątrz znacznika otwierającego może znajdować się atrybut, a atrybut wewnątrz siebie wartość tego atrybutu. Między znacznikami znajduje się jeszcze zawartość elementu, która najczęściej jest wyświetlana w oknie przeglądarki⁵.

CSS – (Cascading Style Sheets) – to język programowania używany do opisu wyglądu i formy prezentacji strony internetowej. Jego głównym celem jest oddzielenie wcześniej wspomnianej struktury od jej stylu wizualnego. CSS można traktować jako zbiór reguł, które definiują, jak poszczególne elementy na stronie powinny wyglądać. Reguły te są tworzone ręcznie przez programistów lub przez internetowe edytory dedykowane stylizacji stron internetowych. Dzięki temu językowi, strony internetowe są bardziej dostępne, elastyczne i łatwe w modyfikacji⁶.

Bootstrap – jest to popularny framework CSS, który oferuje gotowe komponenty i układy, ułatwiając tworzenie responsywnych stron internetowych. Zawiera zestaw narzędzi do projektowania interfejsów użytkownika, takich jak przyciski, formularze, nawigacje i siatki. Bootstrap pozwoli na szybkie i efektywne tworzenie funkcjonalnych i atrakcyjnych interfejsów użytkownika, które będą dobrze wyglądać na różnych urządzeniach⁷.

JavaScript – jest to wieloplatformowy, obiektowy język skryptowy, który odpowiada za funkcjonalność każdej strony internetowej. JS w szczególności służy do złożonych animacji, klikalnych przycisków, menu rozwijanych itp. Istnieją również bardziej zaawansowane wersje JavaScript 'a działające po stronie serwera, takie jak Node.js, które umożliwiają dodanie większej funkcjonalności do witryn internetowych. JavaScript zawiera w sobie standardową bibliotekę obiektów, takich jak Array, Date i Math oraz podstawowy zestaw elementów języka programowania jak np. struktury, operatory, instrukcje i pętle. Po stronie klienta język ten rozszerza podstawę zadeklarowaną wcześniej przy użyciu HTML 'a, dostarczając obiekty do kontrolowania przeglądarki i jej modelu DOM (Document Object Model). Na przykład, rozszerzenia po stronie klienta, pozwalają aplikacji umieszczać elementy w formularzach HTML i reagować na zdarzenia użytkownika, takie jak kliknięcia myszki, wprowadzanie danych i nawigacja po stronach⁸.

⁵ UDI Group, <https://udigroup.pl/blog/jazyk-html-co-to-jest-do-czego-sluzy-jak-wyglada/#co-to-jest-html>, 15.06.2024.

⁶ Kompan.pl, <https://kompan.pl/co-to-jest/css/>, 15.06.2024.

⁷ Bootstrap, <https://getbootstrap.com/docs/5.3/getting-started/introduction/>, 15.06.2024

⁸ Mozilla, <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction?retiredLocale=pl>, 15.06.2024.

3.7 Backend

Mówiąc o Backend ‘zie mówimy inaczej o zapleczu serwerowym, czyli funkcjonalności w tym wypadku strony internetowej z serwerem, bazą danych i obszernymi danymi API serwisów internetowych. W przeciwnieństwie do frontendu, jak nazwa również wskazuje, Backend ‘em nazywamy funkcjonalność niewidoczną dla ludzkiego oka, zaraz po otworzeniu strony internetowej. Ta część zajmuje się dostępami i połączeniami, które tworzą logikę biznesową oraz komunikację z innymi systemami opisanymi poniżej⁹.

PostgreSQL – jest to zaawansowany, opensourcowy system zarządzania relacyjnymi bazami danych. Został on doceniony za zgodność ze standardami SQL, rozszerzalność oraz niezawodność. Obsługuje on złożone operacje, takie jak transakcje ACID, replikacja, indeksowanie pełnotekstowe, oraz zapytania w językach proceduralnych. Jest to często wybierane rozwiązanie bazodanowe w małych aplikacjach po duże rozwiązania hybrydowe.¹⁰

AWS S3 (Simple Storage Service) – to usługa chmurowa firmy Amazon, służąca do przechowywania i zarządzania plikami aplikacji. W kontekście serwisu muzycznego zostanie wykorzystany do przechowywania i udostępniania plików audio przesyłanych przez użytkowników. Zapewnia wysoką dostępność, skalowalność oraz bezpieczeństwo przechowywanych w nim danych. Dzięki integracji z innymi usługami AWS, umożliwia szybkie dostarczenie treści oraz optymalizację kosztów przechowywania. W aplikacji streamingowej S3 pełni kluczową rolę w zarządzaniu biblioteką utworów muzycznych i ich dystrybucji do użytkowników¹¹.

3.8 Serwer

Node.js – opensourcowy, darmowy i multiplatformowa JavaScriptowe środowisko pozwalające na tworzenie aplikacji internetowych, serwerów, skryptów itd. Aplikacje działają w jednym procesie, bez tworzenia wątków dla każdego zapytania. Poprzez swoje zróżnicowane i bardzo rozwinięte biblioteki niepotrzebne jest skakanie między platformami, ponieważ wszystko co potrzebne jest już zawarte w node. Podczas wykonywania operacji takich jak np. odczyty z sieci, node.js nie blokuje wątku i nie marnuje cykli procesora czekając na odpowiedź, natomiast wznowi akcję gdy odpowiedź zostanie już uzyskana.¹²

Express – jest to bardzo popularny framework dla Node.js, który ułatwia tworzenie aplikacji internetowych i API. Jest to elastyczny i lekki zestaw narzędzi, który pozwala na szybkie budowanie serwerów HTTP. Dzięki niemu programiści mogą skupić się na logice aplikacji, a nie na szczegółach związanych z konfiguracją serwera. Express oferuje proste metody do obsługi tras, middleware, a także zapytań HTTP i to dzięki niemu rozwój aplikacji jest bardziej zorganizowany i łatwiejszy.¹³

3.9 Technologie deweloperskie

Visual Studio Code – jest edytorem kodu źródłowego stworzonym przez firmę Microsoft, który jest szeroko używany przez programistów ze względu na swoje funkcje, takie jak podświetlanie składni, autouzupełnianie, debugowanie, zintegrowany terminal czy rozmaite wtyczki upraszczające pracę w systemie. VS code (potocznie nazywany) wspiera wiele języków programowania i ma bogaty ekosystem rozszerzeń. Platforma ta została użyta jako główny edytor kodu, jego edycji, a także debugowania kodu frontendowego i backendowego.¹⁴

Git – jest rozproszonym systemem kontroli wersji, który pozwala na śledzenie zmian w kodzie źródłowym, współprace z innymi programistami i zarządzanie historią projektu. Git umożliwia tworzenie gałęzi (branch) i

⁹ Semcore, <https://semcore.pl/backend-co-to-jest-i-o-czym-wiedziec/>, 15.06.2024.

¹⁰ Postgresql, <https://www.postgresql.org.pl/>, 27.01.2025.

¹¹ Amazon S3, <https://aws.amazon.com/s3/>, 22.02.2025.

¹² Nodejs.org, <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>, 26.01.2025.

¹³ Express.js, <https://expressjs.com/en/guide/routing.html>, 22.02.2025.

¹⁴ Visual Studio Code, <https://code.visualstudio.com/docs>, 22.02.2025.

scalanie (merge) kodu, co ułatwia pracę nad różnymi funkcjonalnościami równocześnie. Git w projekcie będzie pełnił miejsce systemu kontroli wersji¹⁵.

Docker – to narzędzie służące do tworzenia, uruchamiania i zarządzania kontenerami. Kontenerami nazywamy izolowane środowisko, które zawiera aplikację wraz z jej wszystkimi zależnościami. Kontenery pozwalają na uruchomienie aplikacji w sposób spójny, niezależnie od środowiska, w którym działają. Docker pozwala na izolację aplikacji, co zwiększa bezpieczeństwo i ułatwia zarządzanie zależnościami. Pozwala on na stworzenie łatwego i identycznego środowiska programistycznego i testowego, który ułatwia pracę w zespole. Obrazy Dockera można uruchomić na dowolnej maszynie, która ma tylko zainstalowanego dockera¹⁶.

3.10 Wzorce projektowe

Model-View-Controller (MVC) – jest to architektoniczny wzorzec projektowy stosowany do organizacji kodu w aplikacjach webowych. Wzorzec ten dzieli aplikację na trzy warstwy: Model, czyli logika biznesowa i operacje na bazie danych, View, czyli interfejs użytkownika oraz Controller, czyli obsługa żądań i komunikacja między modelem, a widokiem. Dzięki temu wzorcowi aplikacja jest bardziej modularna, łatwiejsza w utrzymaniu i testowaniu.¹⁷

Singleton – Jest to wzorzec zapewniający, że dana klasa ma tylko jedną instancję w całej aplikacji i umożliwia globalny dostęp do niej. W aplikacji został zastosowany do zarządzania połączeniem z bazą danych, co zapobiega wielokrotnemu tworzeniu połączeń i zwiększa wydajność¹⁸.

¹⁵ Git, <https://git-scm.com/about/branching-and-merging>, 22.02.2025.

¹⁶ Docker, <https://docs.docker.com/get-started/docker-overview/>, 22.02.2025.

¹⁷ MVC Pattern, <https://www.geeksforgeeks.org/mvc-design-pattern/>, 22.02.2025.

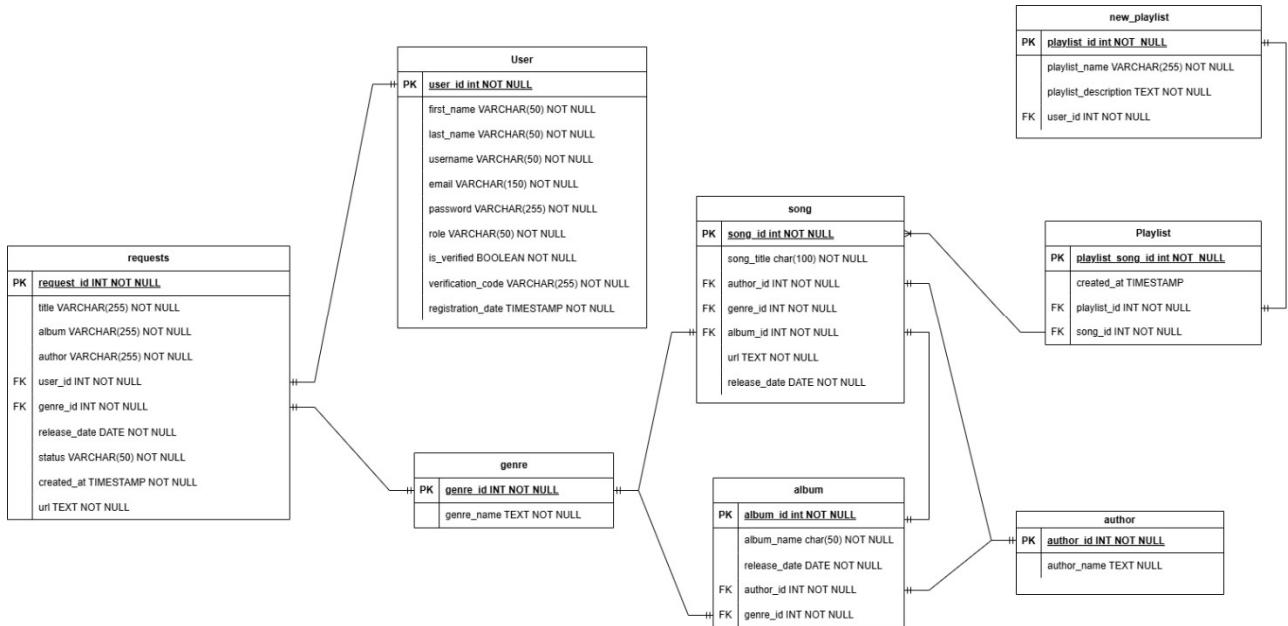
¹⁸ Singleton, <https://refactoring.guru/design-patterns/singleton>, 22.02.2025.

4. Charakterystyka etapów realizacji części projektowej.

W niniejszym rozdziale zostanie przedstawiony proces implementacji systemu, obejmujący wszystkie kluczowe etapy jego tworzenia. Pierwszym krokiem jest zaprojektowanie i stworzenie odpowiedniej struktury danych, która umożliwi efektywne przechowywanie oraz zarządzanie informacjami w bazie danych. Następnie zostanie omówiona budowa backendu, który jest odpowiedzialny za logikę biznesową aplikacji, komunikację z bazą danych oraz obsługę żądań użytkowników. Kolejnym etapem będzie implementacja warstwy frontendowej, która zapewni użytkownikom intuicyjny i funkcjonalny interfejs do interakcji z systemem. Ostatnim etapem jest testowanie aplikacji, które pozwoli na wykrycie ewentualnych błędów oraz optymalizację działania systemu, aby zapewnić jego stabilność i wydajność.

4.1 Etap 1: Stworzenie struktury danych.

Rys. 6. Schemat encji bazy danych



Źródło: Opracowanie własne

4.1.1 Utworzenie bazy danych PostgreSQL wraz z tabelami.

- **User**: przechowująca dane o użytkownikach zarejestrowanych w serwisie takie jak: id, nazwa użytkownika, adres mailowy, hasło, rola (czy użytkownik ma prawa zwykłego użytkownika czy również prawa administratora), imię, nazwisko, czy konto zostało zweryfikowane przez pocztę email, kod uwierzytelniający do weryfikacji użytkownika, data rejestracji użytkownika.
- **Requests**: tabela stworzona w celu weryfikacji danych przesyłanych przez użytkowników. Administrator odczytuje nowe wnioski o dodanie utworu użytkownika do bazy, wpierw w postaci zgłoszenia. Tabela requests posiada pola takie jak: identyfikator zgłoszenia, tytuł nowego utworu, nazwa albumu, autor utworu, identyfikator użytkownika wnioskującego o dodanie nowego utworu, gatunek utworu, data wydania dzieła, status edytowany przez administratora (może być: Zaakceptowany, W trakcie lub Odrzucony), data stworzenia zgłoszenia, ścieżka do pliku mp3 utworu.
- **Songs**: tabela songs jest to zbiór zaakceptowanych i edytowanych przez administratora danych o przesłanym wcześniej przez użytkownika wniosku. W tej tabeli znajdują się utwory, które użytkownicy mogą wyszukać w aplikacji, a następnie dodać je do własnych playlist. Zawiera pola takie jak: nowy identyfikator piosenki (różniący się od tego z tabeli requests), tytuł utworu, album dodany wcześniej przez

administratora aplikacji w procesie sprawdzenia wniosku, data wydania dzieła, autor, ścieżka do pliku, identyfikator gatunku.

- Album: posiadająca pole identyfikatora albumu, jego twórcy, data wydania albumu, sprawdzania przez administratora systemu, a także tytuł zbioru wielu utworów.
- Author: przechowuje indeks autora, a także jego nazwę.
- Genre: dane gatunku muzycznego takie jak identyfikator i nazwa.
- New_playlist: tabela ta zawiera wszystkie niezbędne informacje o playlıście stworzonej przez użytkownika: identyfikator, tytuł playlisty, opis którym twórcą playlist może się dzielić z innymi użytkownikami, a także identyfikator użytkownika który stworzył playlist
- Playlist: siostrzana tabela ‘new_playlist’, posiadająca swój własny identyfikator, jak i ten od swojej siostry, informacja kiedy został ten zbiór utworzony, jak i identyfikator dodanych piosenek do zbioru.

4.2 Etap 2: Budowa backendu

Backend w tej aplikacji pełni kluczową rolę w zarządzaniu logiką biznesową. Przechowuje on dane oraz integruje aplikację z zewnętrznymi usługami, takimi jak AWS S3. Zadaniem backendu jest obsługa zapytań frontendowych w tym tworzeniem nowych playlist, pobieranie ich szczegółowych informacji oraz zarządzanie piosenkami i wnioskami przynależącymi do tych playlist. Dane użytkowników, a następnie playlist, utworów i wniosków przechowywane są w bazie danych PostgreSQL, a backend zapewnia ich wydajne przetwarzanie.

Backend udostępnia wszelkie Endpointy API, które umożliwiają realizację funkcjonalności, takich jak wyszukiwanie piosenek, dodawanie utworów do playlist, czy pobieranie danych o aktualnie zalogowanym użytkowniku. Obsługuje również mechanizm autoryzacji w serwisie. Weryfikuje tożsamość użytkowników używając do tego JWT. Ważnym jego aspektem jest zarządzanie plikami audio i przechowywanie je w S3 bucket, jak również streamowanie ich po stronie klienta.

Zaimplementowana logika backendu pozwala na filtrowanie danych. Podczas wyszukiwania utworów w czasie rzeczywistym lub pobieranie playlist przypisanych w bazie do konkretnego użytkownika. W razie błędów backend zwraca odpowiednie komunikaty, które są wyszczególnione dla każdego endpointu.

Backend jest zarówno zgodny z normami RESTful API, co zapewnia łatwość integracji i skalowalność aplikacji. Zapewnia również wydajną obsługę równoczesnych zapytań i kontrolę dostępu w zależności od uprawnień użytkownika.

4.2.1 Spis endpointów backendu.

- Endpointy do obsługi użytkowników:
 - GET: /verify-email: tworzenie tokenu uwierzytelniającego konto użytkownika,
 - GET: /dashboard: weryfikacja roli użytkownika, dostęp do panelu sterowania wyłącznie dla administratora,
 - GET: /logout: wylogowanie użytkownika, usunięcie sesji z ciasteczką,
 - GET: /api/users: filtrowanie użytkowników według ustawień frontendu, wyświetlanie wszystkich zweryfikowanych użytkowników,
 - GET: /api/current-user: sprawdza sesję autoryzując użytkownika przez JWT, aby wyświetlić jego wszystkie informacje, playlisty w widoku głównym użytkownika
 - DELETE: /user/:id: usuwanie użytkownika o podanym indeksie użytkownika,
 - DELETE: /delete-unverified: usuwanie niezweryfikowanych użytkowników w celu zwolnienia miejsc w bazie danych, a także możliwość ponownego zarejestrowania użytkownika. Konta te są usuwane co godzinę,
 - DELETE: /api/users/:id: usuwanie użytkownika dostępny wyłącznie dla administratora systemu zlokalizowane w profilu użytkownika z podglądu administratora,

POST: /login: logowanie użytkownika, odszyfrowywanie hasła po stronie serwera w celu uwierzytelnienia, tworzenie JWT w ciasteczkach do stworzenia sesji logowania,

POST: /register: do rejestracji użytkownika w serwisie. W pierwszych krokach sprawdzamy email i nazwę użytkownika czy takie konto w systemie już nie istnieje. Jeżeli konto z podanym adresem email istnieje, ale użytkownik nie jest zweryfikowany, usuwa zapisek w bazie, aby dodać go na nowo i wysłać email z kodem weryfikacyjnym ponownie. Następnie hasło podane przez użytkownika jest hashowane i tworzony jest kod weryfikacyjny przy pomocy JWT. Te dane są umieszczane w bazie i na ich podstawie wysyłany jest mail na skrzynkę mailową podaną przez użytkownika. Generowana jest treść tego maila łącznie z linkiem weryfikacyjnym. Na samym końcu obsługa błędów. Kod poniżej.

Rys. 7. Backend – Endpoint register

```
app.post('/register', async (req, res) => {
  const { username, email, password, role, first_name, last_name,
    verification_code, is_verified } = req.body;

  try {
    const checkUsername = await pool.query('SELECT * FROM users WHERE username = $1',
      [username]);
    const existingUser = await pool.query('SELECT * FROM users WHERE email = $1',
      [email]);

    if(checkUsername.rowCount > 0){
      console.log('This username already exist');
      return;
    }

    if(existingUser.rowCount > 0){
      const user = existingUser.rows[0];

      if(!user.is_verified){
        await pool.query('DELETE FROM users WHERE email = $1', [email]);
        console.log('Record deleted. Try to register.');
      }else {
        return res.status(400).send('Account with those credentials already exists.');
      }
    }
  }

  const hashedPassword = await bcrypt.hash(password, 10);

  const verificationToken = jwt.sign({email: email}, SECRET_KEY, {expiresIn: '1h'});
  console.log(verificationToken);

  const result = await pool.query(
    `INSERT INTO users (username, email, password, role, first_name, last_name,
    verification_code, is_verified) VALUES($1, $2, $3, $4, $5, $6, $7, $8) RETURNING *`,
    [username, email, hashedPassword, role, first_name, last_name,
    verificationToken, false]
  );
  const verificationLink = `http://localhost:3000/verify-email?token=${verificationToken}`;
  await transporter.sendMail({
    from: {
      name: 'Music Player',
      address: process.env.USER
    },
    to: email,
    subject: 'Verify your email',
    html: `<p>Click the email below to verify your account: </p><a href="${verificationLink}">${verificationLink}</a>`;
  });

  res.status(200).send(`Registration successful! Please check your email to verify your account`);

} catch (err){
  console.error('Error registering user:', err);
  res.status(500).send('An error occurred during registration');
}
});
```

Źródło: Opracowanie własne

- Endpointy do obsługi wniosków:

GET: /api/requests: filtrowanie i wyświetlanie wszystkich wniosków we frontendzie. Na początku endpointu sprawdzana jest rola, aby to tylko administrator mógł przeglądać wnioski i je edytować.

GET: /api/genres: wyświetlanie wszystkich gatunków muzycznych

GET: /api/requests/:id/metadata: wyświetla wszystkie informacje o wybranym wniosku

GET: /api/requests/:id/audio: przekazuje URL piosenki do odtwarzacza w celu sprawdzenia przez administratora czy plik nadaje się do użytku w aplikacji, a także w celu sprawdzenia wszystkich informacji, aby następnie dodać utwór do tabeli song,

GET: /api/check-album-author: Endpoint sprawdza czy album i autor nie został już dodany wcześniej do tabeli album i author,

GET: /add-album-author-form: przekazanie parametrów albumu i autora,

GET: /api/request-count: zlicza wnioski których status jest ‘w trakcie’,

GET: /api/albums:album_name: wyszukiwanie albumu poprzez jego nazwę,

GET: /api/songs/check: sprawdzenie czy piosenka o takim samym tytule i albumie nie istnieje już w bazie,

POST: /api/requests: tworzenie nowego wniosku o dodanie utworu. Endpoint wykonuje również dodatkowe czynności jakimi są przesyłanie plików lokalnie za pomocą file stream'a, a także przesyłanie je później do s3Bucketu.

POST: /api/add-author: dodanie nowego autora do bazy danych,

POST: /api/add-album: dodanie nowego albumu do bazy danych,

POST: /api/add-song: przekazanie informacji z requests do song w celu dodania nowego utworu,

DELETE: /api/clear-requests: usuwanie wszystkich dostępnych wniosków

PUT: /api/requests/:id, nadpisanie/ edytowanie wniosku przez administratora w celu późniejszego dodania jego finalnej wersji do tabeli song,

- Endpointy do obsługi playlist:

GET: /api/user/playlists: wyświetla wszystkie playlisty użytkownika o przekazanym identyfikatorze,

GET: /api/playlist/:playlistId/songs: wyświetlenie wszystkich dostępnych utworów zawartych w playliście,

GET: /api/playlist/:playlistId/stream: wyświetlenie wszystkich dostępnych utworów, a także umożliwia ich streamowanie w aplikacji,

POST: /api/playlists: dodanie nowej playlisty do dwóch tabel. Do tabeli ‘new_playlist’, która przechowuje informacje o playliście, a także do tabeli ‘playlist’, która przechowuje piosenki,

POST: /api/playlists/:playlistId:songs: dodanie utworu do wybranej playlist,

DELETE: /api/user/playlist/delete: usuwanie playlisty przez użytkownika,

- Endpointy do obsługi streamingu:

GET: /api/songs: wyświetlenie wszystkich dostępnych utworów z tabeli song,

GET: /api/search/songs: endpoint pozwalający na dynamiczne wyszukiwanie utworów w serwisie,

GET: /api/songs/:song_id: wspomiany Endpoint jest najistotniejszym punktem backendu, dlatego chciałbym rozwinąć jego działanie. Początkiem jest przekazanie parametru song_id do endpointu. Następnie występuje komunikacja z bazą czy utwór o takim identyfikatorze istnieje. Odpowiedni komunikat jest wyświetlany w momencie nie znalezienia ID. Po weryfikacji z bazą danych następuje połączenie z S3. Tworzone są nowe zmienne, które służą jako parametry przy zgłoszeniu do S3Bucketu, a także samo wywołanie. Plik dostarczony z S3 zostaje umieszczony w folderze temp aplikacji, zostaje on streamowany, a następnie usunięty pod koniec odtwarzania. Gdy już plik znajduje się w źródle odtwarzacza jest on streamowany. Kończąc cały Endpoint zostaje tylko wywoływanie komunikatów w przypadku błędów aplikacji. End point przedstawiony poniżej. Kod poniżej.

Rys. 8. Backend – Endpoint streamingu utworu

```
app.get('/api/songs/:song_id', async (req, res) => {
  const { song_id } = req.params;

  try{
    const songQuery = await pool.query('SELECT * FROM song WHERE song_id = $1', [song_id]);

    if(songQuery.rows.length === 0 ){
      return res.status(404).json({error: 'Song not found'});
    }

    const song  = songQuery.rows[0];
    const s3Key = song.url.split('.com/')[1];
    const fileName = path.basename(s3Key);
    const localFilePath = path.join(__dirname, 'temp', fileName);
    const getObjectParams = {
      Bucket: bucketName,
      Key: s3Key,
    };

    const command = new GetObjectCommand(getObjectParams);
    const response = await s3.send(command);

    const writeStream = fs.createWriteStream(localFilePath);
    response.Body.pipe(writeStream);

    writeStream.on('finish', () => {
      res.setHeader = ('Content-Type', allowedMimeType);
      const readStream = fs.createReadStream(localFilePath);

      readStream.on('open', () =>{
        readStream.pipe(res);
      })
      readStream.on('error', (err) =>{
        console.error('Error stream the file: ',err);
        res.status(500).json({error: 'Failed to stream file'});
      });

      readStream.on('close', () => {
        fs.unlink(localFilePath, (err) => {
          if (err){
            console.error('Error deleting temporary file: ', err);
          };
        });
      });
    });
    writeStream.on('error', (err) => {
      console.error('Error writing file to local storage: ', err);
      res.status(500).json({ error: 'Failed to save file locally'});
    });
  } catch (err) {
    console.error('Error fetching song: ', err);
    res.status(500).json({error: 'Internal server error'});
  }
});
```

Źródło: Opracowanie własne

- Endpointy inne i funkcje użyte w backendzie:

verifyAdmin() - Funkcja sprawdzająca rolę użytkownika. Funkcja ta jest używana w licznych miejscach do których dostęp powinien mieć tylko i wyłącznie administrator. Funkcja ta dekoduje JWT zapisany w ciasteczkę i sprawdza, czy rola w bazie danych użytkownika to ‘admin’. Jeśli tak nie jest wysyła status 401 Unauthorized. Natomiast jeśli rola się zgadza, pozwala przejść administratorowi dalej. Kod poniżej.

Rys. 9. Backend - Funkcja do weryfikacji czy użytkownik jest administratorem

```

46  function verifyAdmin(req, res, next){
47    const token = req.cookies.auth_token;
48
49    if(!token){
50      return res.status(401).send('Unauthorized');
51    }
52
53    try {
54      const decoded = jwt.verify(token, SECRET_KEY);
55
56      if(decoded.role !== 'admin') {
57        return res.status(403).send('Access denied');
58      }
59
60      next();
61    } catch (err) {
62      console.error('Error verifying admin:', err)
63      res.status(401).send('Unauthorized');
64    }
65  }
66

```

Źródło: Opracowanie własne

Ustawienia AWS S3Bucket, a także przesyłanie plików lokalnie w celu ich przesłuchiwanego, ale wpierw w uploadowaniu ich na serwer AWS. W pierwszej części czyli zmiennej S3 tworzymy nowy S3 przy użyciu klasy S3Client z biblioteki AWS SDK. Wykorzystuje on dane takie jak region i klucze dostępu, które są pobierane z zmiennych środowiskowych (process.env) w celu uwierzytelnienia i komunikacji z usługą S3. Zmienna bucketName zarówno jest częścią S3 przechowuje ona nazwę bucketu.

Zmienna storage i funkcja diskStorage definiuje lokalny sposób przechowywania plików. W destination ustawiamy lokalny folder /uploads , w którym będą znajdować się gotowe do przesłania pliki na serwer jeszcze przed zatwierdzeniem frontendowego formularza. W filename definiujemy unikalną nazwę dla każdego pliku, poprzez Date.now() i losowy numer.

W końcowej części konfigurujemy samo przesyłanie pliku za pomocą multer. Storage zdefiniowane w poprzednim punkcie określa przechowywanie lokalne pliku. fileFilter filzuje pliki, aby sprawdzić rozszerzenie i dozwolony typ MIME. W przeciwnym wypadku zwracany będzie błąd ‘Invalid file type’. Kod poniżej.

Rys. 10. Backend - Połączenie pomiędzy serwerem a serwisem AWS.

```
73 const s3 = new S3Client({
74   region: process.env.AWS_REGION,
75   credentials: {
76     accessKeyId: process.env.AWS_ACCESS_KEY_ID,
77     secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
78   },
79 });
80
81 const bucketName = process.env.AWS_BUCKET_NAME;
82
83 const storage = multer.diskStorage({
84   destination: (req, file, cb) => {
85     cb(null, "uploads/");
86   },
87   filename: (req, file, cb) => {
88     const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
89     cb(null, uniqueSuffix + path.extname(file.originalname));
90   },
91 });
92
93 const upload = multer({
94   storage,
95   fileFilter: function (req, file, cb) {
96     const allowedMimeType = ['audio/mpeg', 'audio/wav', 'audio/ogg'];
97     if(allowedMimeType.includes(file.mimetype)){
98       cb(null, true);
99     } else {
100       cb(new Error('Invalid file type. Only audio files allowed.'))
101     }
102   },
103 });
104
```

Źródło: Opracowanie własne

Konfiguracja nodemailera w zmiennej transporter. Nodemailer ma na celu wysyłanie maila serwisem gmail. Mail jest potrzebny do uwierzytelnienia adresu mailowego użytkownika. Kod poniżej.

Rys. 11. Backend - Konfiguracja nodemailer'a

```
125 const transporter = nodemailer.createTransport({
126   service: 'gmail',
127   host: 'smtp.gmail.com',
128   auth: {
129     user: process.env.MAIL_USER,
130     pass: process.env.MAIL_PASSWORD
131   }
132 })
133
```

Źródło: Opracowanie własne

4.3 Etap 3: Implementacja frontendu

Frontend w tej aplikacji zajmuje się obsługą interfejsu użytkownika. Pozwala on na interakcje z backendem umożliwiając interakcję między nimi. Odpowiada on za wyświetlanie dynamicznych danych, takich jak playlisty, użytkownicy, utwory czy wnioski o dodanie nowych utworów. Robi to za pomocą zapytań do odpowiednich endpointów API (Application Programming Interface). Użytkownik, za pomocą formularzy, jest w stanie tworzyć nowe playlisty, dodawać im opis, modyfikować ich zawartość, co realizowane jest dzięki komponentom HTML, CSS i JavaScript. Dzięki mechanizmowi LiveSearch frontend jest w stanie na rzeczywiste wyszukiwanie utworu i filtruje dane na podstawie wprowadzonego tekstu.

Odtwarzacz jest tutaj kluczowym elementem frontendowej części aplikacji. Zapewnia on ciągłość odtwarzania piosenek. Został on zaprojektowany w taki sposób, aby utwory były pobierane na podstawie identyfikatora playlistId, a następnie streamowane przez integralność S3 z backendem. Interfejs jest responsywny, przez co aplikacja działa zarówno dobrze na urządzeniach mobilnych, jak i na komputerach. Frontend w oparciu o backend zarządza również jaki użytkownik ma dostęp do jakich treści. Zastosowanie roli administratora i funkcji verifyAdmin() pozwala na administrację użytkownikiem od samej rejestracji do wylogowania z aplikacji.

4.3.1 Interfejs użytkownika.

- Strona główna

Wyświetla podstawowe informacje o zalogowanym użytkowniku takie jak imię, nazwisko, nazwa użytkownika. Posiada zakładkę moje playlisty, dynamiczne wyszukiwanie piosenek, a także dodaj nową playlistę.

Funkcja checkCurrentUser() sprawdza, czy użytkownik jest aktualnie zalogowany i dynamicznie aktualizuje interfejs strony w zależności od statusu autentykacji. Na początek wysyła żądanie GET od endpointu. Jeśli odpowiedź jest pozytywna (status 200) to: wyświetla sekcję informacji o użytkowniku, dopasowuje menu, aby ukazała się opcja do wylogowania, dodania playlisty, dodania utworu. Gdy użytkownik przy okazji okaże się administratorem również ukazuje się panel kontrolny administratora. Natomiast gdy zwrócony zostanie błąd autentykacji użytkownik próbujący się zalogować otrzyma przycisk do logowania i rejestracji. Kod poniżej.

Rys. 12. Frontend - Strona główna. Sprawdzanie czy użytkownik jest zalogowany

```
69  |     async function checkCurrentUser(){
70  |   try{
71  |     const response = await fetch('/api/current-user', {
72  |       method: 'GET',
73  |       headers: {
74  |         'Content-Type': 'application/json'
75  |       },
76  |       credentials:"include"
77  |     });
78  |
79  |     if(response.ok){
80  |       const user = await response.json();
81  |
82  |       document.getElementById('user-info').style.display = 'block';
83  |       document.getElementById('user-name').textContent = `${user.first_name} ${user.last_name}`;
84  |       document.getElementById('user-username').textContent = user.username;
85  |       document.getElementById('user-email').textContent = user.email;
86  |       document.getElementById('user-role').textContent = user.role;
87  |
88  |       document.getElementById('dashboard-username').innerHTML = `<a href="">Hello, ${user.first_name}</a>`;
89  |
90  |       document.getElementById('dashboard-link').style.display = user.role === 'admin' ? 'inline' : 'none';
91  |       document.getElementById('upload-file').style.display = 'inline';
92  |       document.getElementById('logout-link').style.display = 'inline';
93  |
94  |       document.getElementById('playlist-button').style.display = 'block';
95  |
96  |       document.getElementById('playlists_containter').style.display = 'block';
97  |       document.getElementById('playlist-h1').style.display = 'block';
98  |       currentUserID = user.user_id;
99  |       fetchPlaylists(currentUserID)
100 |     }else {
101 |       document.getElementById('auth-buttons').style.display = 'block';
102 |       document.getElementById('login-link').style.display = 'inline';
103 |       document.getElementById('register-link').style.display = 'inline';
104 |     }
105 |   }catch (err) {
106 |     console.error('Error checking user:', error);
107 |     document.getElementById('auth-buttons').style.display = 'block';
108 |     document.getElementById('login-link').style.display = 'inline';
109 |     document.getElementById('register-link').style.display = 'inline';
110 |   }
111 | }
112 | }
```

Źródło: Opracowanie własne

- Wyszukiwarka utworów

Umożliwia wyszukiwanie utworów po tytule, albumie i wykonawcy jednocześnie. Obsługuje live search, co oznacza, że utwory są wyświetlanie dynamicznie względem wpisanej frazy.

Wyszukiwanie na żywo z obsługą playlist. Funkcja addEventListener nasłuchiwa wpisywania w polu wyszukiwania, używa opóźnienia, aby uniknąć nadmiarowych zapytań przy szybkim pisaniu. Wysyła zapytania z frazą wyszukiwania, a także obsługuje wyniki. Wyświetla znalezione utwory przez displaySearchResult. Pokazuje komunikat ‘No songs found’ dla statusu 404, czyli gdy nie znaleziono dopasowania do wprowadzonego zapytania. Druga funkcja, czyli displaySearchResult renderuje wyniki w kontenerze. Umieszczony jest tam tytuł, wykonawca i album, a także przycisk ‘Add to playlist’, po którego kliknięciu ukazuje się okno typu ‘Modal’ z zapytaniem, do której playlisty użytkownik chce dodać utwór.

Rys. 13. Frontend - Funkcja odpowiedzialna za Live-Search

```
256  searchContainer.addEventListener('input', async (event) => {
257    const query = event.target.value.trim();
258
259    if(searchTimeout){
260      clearTimeout(searchTimeout);
261    }
262
263    if(query === ''){
264      document.getElementById('live-search-results').innerHTML = '';
265      return;
266    }
267
268    searchTimeout = setTimeout(async () => {
269      try{
270        const response = await fetch(`/api/search/songs?query=${encodeURIComponent(query)}`);
271
272        if(response.ok) {
273          const songs = await response.json();
274          displaySearchResult(songs);
275        } else if (response.status === 404){
276          document.getElementById('live-search-results').innerHTML = '<p>No songs found</p>';
277        }else {
278          throw new Error('Failed to fetch search results');
279        }
280      } catch (err) {
281        console.error('Error searching for songs:', err);
282        alert('An error occurred while searching for songs');
283      }
284    })
285  })
286
```

Źródło: Opracowanie własne

- Tworzenie playlisty

Formularz umożliwiający dodanie nowej playlisty. Po dodaniu playlisty dodają się opcje edycji playlisty względem upodobań użytkownika.

Funkcja zbiera informacje z formularza dotyczące nowej playlisty. Za pomocą żądania metodą POST tworzy zapytanie SQL dodające nowy rekord do bazy danych. Strona również informuje użytkownika, czy proces został zakończony pozytywnie, lub w przeciwnym wypadku, co poszło nie tak. Kod poniżej.

Rys. 14. Frontend - Tworzenie nowej playlisty.

```
document.getElementById('new-playlist-form').addEventListener('submit', async (event) => {
  event.preventDefault();

  const playlist_name = document.getElementById('playlist-name').value;
  const playlist_description = document.getElementById('playlist-description').value;

  const newUser = {
    playlist_name,
    playlist_description,
    user_id: currentUser
  };
  try {
    const response = await fetch('/api/playlists', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(newUser)
    });

    if (response.ok) {
      const data = await response.json();
      alert('Playlist created successfully! Playlist ID: ${data.playlist_id}');
      document.getElementById('new-playlist-form').reset();
      window.location.href = '/';
    } else {
      const error = await response.json();
      alert(`Failed to create playlist: ${error.message}`);
    }
  } catch (err) {
    console.error('Error creating playlist:', err);
    alert('An error occurred while creating the playlist.');
  }
});
```

Źródło: Opracowanie własne

- **Moje playlisty**

Wyświetla listę wszystkich playlist utworzonych przez użytkownika. Po kliknięciu w ‘Go to the player’ Otwiera się odtwarzacz, który pozwala na przesłuchiwanie wybranej składanki.

Funkcja fetchPlaylists przekazuje argument w postaci indeksu użytkownika. Za pomocą żądania GET pobiera wszystkie playlisty użytkownika o podanym indeksie. Funkcja displayPlaylists została stworzona w celu pozyskania wszystkich informacji o wszystkich playlistach użytkownika. Oprócz informacji, takich jak co się w niej znajdują przygotowane są przyciski pozwalające otworzyć składankę, a także ją usunąć w razie potrzeby. Kod poniżej.

Rys. 15. Frontend - Wyświetlanie wszystkich playlist należących do zalogowanego użytkownika.

```

118    async function fetchPlaylists(userId) {
119      try {
120        const response = await fetch(`/api/user/playlists?user_id=${userId}`, {
121          method: 'GET',
122          credentials: 'include',
123        });
124
125        if (response.ok) {
126          const playlists = await response.json();
127          displayPlaylists(playlists);
128        } else {
129          console.error('Failed to fetch user playlists');
130          document.getElementById('playlist-info').innerHTML = 'No playlists';
131        }
132
133      } catch (err) {
134        console.error('Error fetching playlists', err);
135      }
136    }
137
138    function displayPlaylists(playlists) {
139      const container = document.getElementById('playlists_container');
140      container.innerHTML = '';
141
142      playlists.forEach((playlist) => {
143        const playlistDiv = document.createElement('div');
144        playlistDiv.className = 'playlist';
145
146        playlistDiv.innerHTML = `
147          <h3>${playlist.playlist_name}</h3>
148          <p>${playlist.playlist_description}</p>
149
150          <button onclick="fetchSongs(${playlist.playlist_id})">Show Songs</button>
151          <button onclick="deletePlaylist(${playlist.playlist_id})">Delete Playlist</button>
152
153          <!-- Link do player.html z przekazaniem ID playlisty w parametrze -->
154          <a href="player.html?playlistId=${playlist.playlist_id}">Go to Player</a>
155
156        `;
157        container.appendChild(playlistDiv);
158      });
159    }

```

Źródło: Opracowanie własne

- **Odtwarzacz muzyczny.**

Dla odtwarzacza została przeznaczona osobna podstrona, która pozwala na pełne skupienie użytkownika w odsłuchiwanej przez siebie składance muzycznej. Sam odtwarzacz zbudowany jest z listy utworów przekazywanych za pomocą backendu do nowo otworzonej podstrony. Odtwarzacz jest responsywny, co oznacza, że w momencie gdy w playliście jest więcej niż pięć utworów, odtwarzacz nie rozwleka je na całą stronę, tylko zatrzymuje się na pięciu, dla komfortu widoku utworów. Na sam początek jest tworzona tablica piosenek, która zostaje uzupełniona o wszystkie utwory z backendu, aby w kolejnych krokach ułatwić takim funkcjonalnościom, jak odtwarzanie losowe czy odtwarzanie w pętli. Dodaje to również indeksowanie piosenkami, gdyż w momencie, gdy ostatnia piosenka zakończy swe odtwarzanie, odtwarzacz nie zatrzyma się tylko załaduje pierwszą z kolei. Kod poniżej.

Rys. 16. Frontend – Odtwarzacz muzyki

```
118
119     async function fetchPlaylistSongs() {
120         try {
121             const response = await fetch(`/api/playlists/${playlistId}/songs`);
122             if (!response.ok) {
123                 throw new Error('Failed to fetch playlist songs');
124             }
125             songs = await response.json();
126             displaySongList(songs);
127         } catch (error) {
128             console.error('Error fetching playlist songs:', error);
129             songListContainer.innerHTML = `

Error loading playlist.</p>`;
130         }
131     }
132     function displaySongList(songsToDisplay) {
133         if (!songsToDisplay || songsToDisplay.length === 0) {
134             songListContainer.innerHTML = `

No songs available.</p>`;
135             return;
136         }
137
138         if (songsToDisplay.length > 5) {
139             songListContainer.style.maxHeight = '300px';
140             songListContainer.style.overflowY = 'auto';
141         } else {
142             songListContainer.style.maxHeight = '';
143             songListContainer.style.overflowY = '';
144         }
145
146         songListContainer.innerHTML = '';
147         songsToDisplay.forEach((song, index) => [
148             const songItem = document.createElement('div');
149             songItem.classList.add('song-item');
150
151             const originalIndex = songs.indexOf(song);
152
153             songItem.textContent = `${song.title} by ${song.author_name || 'Unknown'}`;
154             songItem.dataset.songIndex = originalIndex;
155
156             songItem.addEventListener('click', () => {
157                 playSongByIndex(originalIndex);
158             });
159
160             songListContainer.appendChild(songItem);
161         ]);
162     }
163 }


```

Źródło: Opracowanie własne

- Odtwarzanie losowe (Shuffle mode) i odtwarzanie w pętli (repeat mode).
Shuffle w aplikacji działa w sposób następujący. Na sam wpierw tworzona jest tablica, która jest uzupełniana wszystkimi utworami składanki. W momencie, gdy shuffle mode jest włączony indeksy tablicy są odtwarzane losowo, a nie 0, 1, 2, 3 Repeat mode polega na odtwarzaniu jednego utworu w pętli. Kod poniżej.

Rys. 17. Frontend – funkcje odpowiedzialne za odtwarzanie losowe i odtwarzanie w pętli.

```

213 |   function nextSong() {
214 |     if (repeatMode) {
215 |       playSongByIndex(currentIndex);
216 |       return;
217 |     }
218 |
219 |     if (shuffleMode && songs.length > 1) {
220 |       let randomIndex = currentIndex;
221 |       while (randomIndex === currentIndex) {
222 |         randomIndex = Math.floor(Math.random() * songs.length);
223 |       }
224 |       playSongByIndex(randomIndex);
225 |     } else {
226 |       if (currentIndex < songs.length - 1) {
227 |         playSongByIndex(currentIndex + 1);
228 |       }
229 |     }
230   }
231   function toggleShuffle() {
232     shuffleMode = !shuffleMode;
233     shuffleButton.textContent = shuffleMode ? 'Shuffle On' : 'Shuffle Off';
234     if (shuffleMode) {
235       shuffleButton.classList.add('active-mode');
236     } else {
237       shuffleButton.classList.remove('active-mode');
238     }
239   }
240   function toggleRepeat() {
241     repeatMode = !repeatMode;
242     repeatButton.textContent = repeatMode ? 'Repeat On' : 'Repeat Off';
243     if (repeatMode) {
244       repeatButton.classList.add('active-mode');
245     } else {
246       repeatButton.classList.remove('active-mode');
247     }
248   }

```

Źródło: Opracowanie własne

- Wyszukiwanie utworów w playliście.
Wyszukiwanie polega na przymierzeniu fraz z utworami należącymi do tablicy.

Rys. 18. Frontend – Wyszukiwanie utworów w playliście

```

260 |
261 |   searchInput.addEventListener('input', (event) => {
262 |     const query = event.target.value.toLowerCase().trim();
263 |     const filteredSongs = songs.filter(song => {
264 |       const titleMatch = song.title?.toLowerCase().includes(query);
265 |       const authorMatch = song.author_name?.toLowerCase().includes(query);
266 |       return titleMatch || authorMatch;
267 |     });
268 |     displaySongList(filteredSongs);
269   });
270

```

Źródło: Opracowanie własne

4.3.2 Interfejs administratora.

Pakiet interfejsu administratora rozszerza opcje zwykłego użytkownika o wymienione poniżej funkcjonalności.

- Zarządzanie kontami użytkowników.

Użytkownik z rolą administratora otrzymuje dodatkowe funkcjonalności, takie jak edycja i moderacja innymi użytkownikami serwisu internetowego. Administrator otrzymuje wejście do dashboardu, w którym jest w stanie wyfiltrować wszystkich użytkowników jak i moderować ich konta. W pierwszych krokach sprawdzana jest rola użytkownika, który otrzymał dostęp do panelu kontrolnego użytkowników. JSONWebToken jest dekodowany jeszcze w backendzie. Następnie ukazuje się sortowanie. Administrator może użyć filtrów względem różnych parametrów, a także najzwyczajniej wyszukać użytkownika. Na samym końcu wszyscy użytkownicy są wyświetlani przez funkcję fetchUsers(); Kod poniżej.

Rys. 19. Frontend – Panel administratora – wypisanie wszystkich użytkowników

```
async function fetchUsers(sortBy = 'user_id', sortOrder = 'asc', searchColumn = '', searchTerm = '') {
    try{
        const response = await fetch(`/api/users?sortBy=${sortBy}&sortOrder=${sortOrder}&searchColumn=${searchColumn}&searchTerm=${searchTerm}` , {
            method: 'GET',
            headers: {
                'Content-Type': 'application/json'
            }
        });

        if(response.ok) {
            const users = await response.json();
            const tableBody = document.querySelector('#users-table tbody');
            tableBody.innerHTML = '';

            users.forEach(user => {
                const row = document.createElement('tr');

                row.innerHTML = `
                    <th>${user.user_id}</th>
                    <th>${user.first_name}</th>
                    <th>${user.last_name}</th>
                    <th>${user.username}</th>
                    <th>${user.email}</th>
                    <th>${user.is_verified ? 'Yes' : 'No'}</th>
                    <th><button onclick = "showProfile(${user.user_id})>Show Profile</button></th>
                `;

                tableBody.appendChild(row);
            });
        } else {
            console.error('Failed to fetch users');
        }
    } catch (err) {
        console.error('Error fetching users:',err);
    }
}
```

Źródło: Opracowanie własne

- Profile wszystkich użytkowników.

Przez przekazanie parametrów administrator może otworzyć profil każdego użytkownika z bazy. Dostęp jest również chroniony przez funkcję checkCurrentUser(), która to umożliwia sprawdzenie kontentu wyłącznie administratorom.

Za pomocą żądania endpointu metodą GET zwarcane są wszystkie informacje z bazy danych z tabeli users. Następnie skrypt tworzy z nich tabelę, gdzie wszystkie informacje są wypisane i dobrze widoczne dla administratora. Kod poniżej.

Rys. 20. Frontend – panel administratora – profil użytkownika

```
async function fetchUserProfile() {
  const params = new URLSearchParams(window.location.search);
  const userID = params.get('id');

  if(!userID){
    document.getElementById('user-details').innerHTML = '<p>User ID not provided.</p>';
    return;
  }

  try{
    const response = await fetch(`/api/users/${userID}`, {
      method: 'GET',
      headers:{
        'Content-Type': 'application/json'
      }
    });

    if(response.ok){
      const user = await response.json();
      document.getElementById('user-details').innerHTML = `

        <p><strong>ID:</strong> ${user.id}</p>
        <p><strong>First Name:</strong> ${user.first_name}</p>
        <p><strong>Last Name:</strong> ${user.last_name}</p>
        <p><strong>Username:</strong> ${user.username}</p>
        <p><strong>Email:</strong> ${user.email}</p>
        <p><strong>Verified:</strong> ${user.is_verified ? 'Yes' : 'No'}</p>
        <p><strong>Account created:</strong> ${user.registration_date}</p>

    `

    }else {
      document.getElementById('user-details').innerHTML = '<p>Failed load the user</p>';
    }
  }catch (err) {
    console.error('Error fetching user profile ',err);
    document.getElementById('user-profile').innerHTML = '<p>An error occurred while loading user details. </p>'

  }
}
```

Źródło: Opracowanie własne

- Administrowanie wnioskami o dodanie nowych utworów

Jednym z bardziej rozbudowanych rozwiązań backendowych, ale i frontendowych tej aplikacji jest moderowanie wnioskami użytkowników. Użytkownik wpierw musi zawnioskować o dodanie nowego dzieła do aplikacji. Następnie w karcie Requests w panelu administratora pojawia się nowy wniosek, który jest w trakcie akceptacji. Administrator po otwarzeniu wszystkich wniosków dostaje możliwość odfiltrowania tych, które są w trakcie akceptacji. Dostępne jest również wiele innych parametrów do sprawdzenia. Administrator odsłuchuje wniosek, który po kliknięciu przycisku ‘PLAY’ zostaje przekazany do źródła odtwarzacza. Po przesłuchaniu to administrator decyduje, co może zrobić z wnioskiem. Moderator może go edytować, aby nadać mu nowy tytuł, zmienić gatunek, datę wydania, album, wykonawcę lub status. Jeśli zdecyduje się na dodanie utworu do zbioru song wywoływane są pewne funkcje sprawdzające bazę danych. Wpierw sprawdza się czy Album i Wykonawca istnieją już w bazie danych. Nowy formularz otwiera się zależnie od wyniku np. jeżeli tylko wykonawca istnieje, a album trzeba dodać otwiera się formularz na dodanie nowego albumu i przeciwnie. Jeżeli album i wykonawca istnieją w bazie sprawdzane jest czy takowa piosenka już nie istnieje w tabeli song. W razie błędów przygotowane są odpowiednie powiadomienia informujące administratora systemu o sprecyzowanych błędach. Gdy wszystkie informacje znajdują się w bazie, wniosek może zmienić swój status na ‘zatwierdzony’ i utwór zostaje dodany do bazy.

Dodawanie utworu do bazy krok po kroku:

- Wpierw sprawdzane jest czy album i wykonawca istnieją.

Funkcja checkAlbumAndAuthor wysyła żądanu do backendu w celu pobrania informacji, czy podane w parametrach album i wykonawca istnieją. Kod poniżej.

Rys. 21. Frontend – sprawdzenie istniejących danych

```
async function checkAlbumAndAuthor(albumName, authorName, requestId) {
    try {
        const response = await fetch(`/api/check-album-author?albumName=${encodeURIComponent(albumName)}&authorName=${encodeURIComponent(authorName)}`);
        const { albumExists, authorExists } = await response.json();

        if(!authorExists || !albumExists){
            openDynamicForm(albumName, authorName, albumExists, authorExists, requestId);
        } else{
            await updateRequest(requestId);
        }
    } catch(err){
        console.error('Error in checking album and author' ,err);
    }
}

document.getElementById('cancel-edit').addEventListener('click', () => {
    document.getElementById('edit-request-container').style.display = 'none';
})
```

Źródło: Opracowanie własne

- Gdy w bazie brakuje danych otwierany jest formularz do wprowadzenia nowych wartości. Funkcja otwiera dynamiczny formularz zawierający, zależne od sytuacji, pola do wprowadzenie przez administratora. Formularz zawiera pola do wprowadzenia nowego albumu i wykonawcy, natomiast zależność czy otwierane są oba, czy tylko jeden, a może żaden znajduje się po stronie backendu. Kod poniżej

Rys. 22. Frontend – funkcja otwierająca panel dodawania nowego albumu i wykonawcy

```
function openDynamicForm(albumName, authorName, albumExists, authorExists, requestId){
    const dynamicFormContainer = document.getElementById('dynamic-form-container');
    dynamicFormContainer.style.display = 'block';

    if(!authorExists) {
        document.getElementById('new-author-section').style.display = 'block';
        document.getElementById('dynamic-author-name').value = authorName;
    }else {
        document.getElementById('dynamic-author-id').value = authorExists.author_id;
    }

    if(!albumExists){
        document.getElementById('new-album-section').style.display = 'block';
        document.getElementById('dynamic-album-name').value = albumName;
    }
}
```

Źródło: Opracowanie własne

- Po zakończeniu zgłoszenia frontend odwołuje się do backendu w celu zaktualizowania nowych danych i dodania ich do bazy danych. Funkcja w tej części wykorzystuje już wcześniej pozyskane dane, czy album lub wykonawca już istnieją. Zależność ta jest sprawdzana w instrukcjach warunkowych i o ile warunek jest spełniony, żądania POST są wywoływanie, aby dodać nowy rekord do bazy danych. Kod poniżej.

Rys. 23. Frontend – odwołanie się do backendu w celu sprawdzenia i dodania danych

```
document.getElementById('dynamic-form').addEventListener('submit', async (event) => {
  event.preventDefault();
  try{
    let authorId = document.getElementById('dynamic-author-id').value;
    let albumId = null;

    if(!authorExists) {
      const authorResponse = await fetch('/api/add-author', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          author_name: document.getElementById('dynamic-author-name').value
        })
      });

      if(!authorResponse.ok){
        throw new Error('Failed to add author');
      }

      const authorData = await authorResponse.json();
      authorId = authorData.author_id;
      document.getElementById('dynamic-author-id').value = authorId;
      console.log('Author added successfully');

    }

    if(!albumExists) {
      const albumResponse = await fetch('/api/add-album', {
        method: 'POST',
        headers: [],
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        album_name: document.getElementById('dynamic-album-name').value,
        release_date: document.getElementById('dynamic-release-date').value,
        genre_id: document.getElementById('dynamic-genre-id').value,
        author_id: document.getElementById('dynamic-author-id').value
      }),
    });
    const albumData = await albumResponse.json();
    albumId = albumData.album_id;
  }

  await updateRequest(requestId);
  dynamicFormContainer.style.display = 'none';
}
```

Źródło: Opracowanie własne

4.4 Etap 4: Testowanie

Przeprowadzanie testów jednostkowych – test jednostkowe tworzy się w celu weryfikacji działania pojedynczych funkcji, metod lub modułów aplikacji. Służą one do wykrywania błędów na wczesnym etapie tworzenia oprogramowania, co pomaga w utrzymaniu wysokiej jakości kodu i zapobieganiu problemów w przyszłości przy kontynuacji. Programiści właśnie dzięki testom jednostkowym są w stanie szybciej identyfikować i zapobiegać błędom. Test również zapobiega bezpieczeństwu aplikacji, umożliwiając automatyczne sprawdzanie poprawności działania kluczowych fragmentów kodu.

Testy jednostkowe endpointów:

Test endpointu current-user.test.js sprawdza zachowanie endpointu w różnych scenariuszach. Pierwsze dwa przypadki testowe weryfikują obsługę błędów. Zwraca 401 gdy brakuje tokenu lub token jest nieprawidłowy. Kolejne dwa przypadki są wywoływane w momencie gdy użytkownik nie istnieje w bazie i ostatni, gdy użytkownik istnieje i wygenerowany token jest prawidłowo dopasowany.

Rys. 24. Test jednostkowy endpointu current-user

```
PASS tests/current-user.test.js
GET /api/current-user
✓ should return 401 if no token is provided (22 ms)
✓ should return 401 if the token is invalid (38 ms)
✓ should return 404 if the user is not found (4 ms)
✓ should return 200 with user data if the token is valid and user exists (2 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.488 s, estimated 1 s
Ran all test suites.
```

Źródło: Opracowanie własne

Test jednostkowy login.test.js sprawdza działanie endpointu POST /login w kilku scenariuszach. Pierwszy weryfikuje poprawne logowanie użytkownika z prawidłowymi danymi, sprawdzając, czy ustawiany jest token i odpowiedź ma status 302 (przekierowanie). Kolejne testy sprawdzają sytuacje błędne jak: logowanie z niepoprawnym hasłem 401, brak użytkownika w bazie 404, a także obsługę błędów serwera 500.

Rys. 25. Test jednostkowy endpointu login

```
PASS tests/login.test.js
POST /login
✓ should log in successfully with valid credentials (28 ms)
✓ should return 401 with invalid credentials (14 ms)
✓ should return 404 if user does not exist (3 ms)
✓ should handle internal server errors (12 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.518 s, estimated 1 s
```

Źródło: Opracowanie własne

Test add-album.test.js sprawdza działanie endpointu POST /api/add-alubm w dwóch jego scenariuszach. Pierwszy test weryfikuje poprawne dodanie albumu do bazy danych przy, sprawdzając przy okazji, czy zwracana jest prawidłowa odpowiedź 200 oraz, czy zapytanie SQL zostało wykonane poprawnie. Drugi test sprawdza obsługę serwera 500, gdy wystąpi problem z bazą danych, upewnia się, że odpowiedź zawiera komunikat ‘Server error’.

Rys. 26. Test jednostkowy endpointu add-album

```
PASS  tests/add-album.test.js
POST /api/add-album
  ✓ should add an album successfully (31 ms)
  ✓ should return 500 on server error (22 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.494 s, estimated 1 s
Ran all test suites.
```

Źródło: Opracowanie własne

Test jednostkowy bazy danych:

Test sprawdza poprawność działania funkcji getAlbums i getAlbumByID, które pobierają albumy z bazy danych. Pierwszy test weryfikuje, czy funkcja getAlbums zwraca poprawną listę albumów, a drugi sprawdza, czy getAlbumById zwraca właściwy album na podstawie podanego indeksu. Dodatkowo, testy obejmują przypadek, gdy album o danym ID nie istnieje oraz sytuację, gdy występuje błąd bazy danych.

Rys. 27. Test jednostkowy bazy danych

```
PASS  tests/db.test.js
Database Queries
  ✓ should fetch all albums (2 ms)
  ✓ should fetch album by ID (1 ms)
  ✓ should handle empty results for getAlbumById (1 ms)
  ✓ should handle database errors (3 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.486 s
Ran all test suites.
```

Źródło: Opracowanie własne

5. Interfejs użytkowników

Interfejs użytkownika (UI) aplikacji odgrywa kluczową rolę w zapewnieniu intuicyjnej i wygodnej obsługi dla użytkownika. W niniejszym rozdziale przedstawione zostaną główne ekrany aplikacji oraz ich funkcjonalność. Ekrany te zostaną pokazane, jako stan faktyczny widoku ekranu aplikacji. Założeniem projektu była funkcjonalność, która w większości skupiła moja uwagę przy wytwarzaniu oprogramowania.

5.1 Widoki użytkownika

5.1.1 Ekran logowania i rejestracji

Te dwa ekrany umożliwiają użytkownikowi logowanie lub rejestracje (zależnie czy mają stworzone konto). Ekran logowania po wpisaniu poprawnych danych użytkownika na zalogowanie do serwisu natomiast ekran rejestracji na dołączenie jako nowy użytkownik.

Proces rejestracji

Użytkownik:

1. Otwieramy stronę. W przypadku rejestracji jest to localhost:3000/register
2. Wpisujemy informację takie jak: Imię, Nazwisko, Nazwa użytkownika, Adres E-mail, Hasło jak i również to hasło musimy powtórzyć.
3. Klikamy w przycisk ‘Register’

Aplikacja:

1. Po kliknięciu w przycisk włącza się frontendowy skrypt sprawdzający. Funkcja nazywa się *passwordRequirementsCheck()*. Sprawdza czy w nazwie użytkownika nie ma znaków specjalnych, czy został podany właściwy adres mailowy, czy pole *Password* i *Confirm Password* są takie same. Następnie sprawdzane są zasady bezpiecznego hasła które są następujące:
 - a. Długość hasła dłuższa niż 8 znaków
 - b. Minimum jedna wielka i mała litera
 - c. Cyfra
 - d. Znak specjalny

W przypadku niespełnienia warunków funkcji, odpowiedni alert pojawia się aby poinformować użytkownika o braku kompatybilności z polityką rejestracji aplikacji.

2. Rekord zostaje dodany do bazy, a jego hasło zostaje zhashowane dla jeszcze większego bezpieczeństwa.
3. Backend sprawdza czy użytkownik nie istnieje już w bazie, a strona wyświetla wiadomość, że użytkownik został zarejestrowany pomyślnie i prosi o uwierzytelnienie. Autoryzacją użytkownika jest wiadomość email na wskazany adres. Wiadomość tworzy nodemailer. W bazi danych, za pomocą JWT (JSONWebToken) tworzony jest specjalny kod uwierzytelniający, który również jest zawarty w linku aktywacyjnym wysłanym do użytkownika.
4. Użytkownik w ciągu godziny powinien wejść w link i zautoryzować konto. Po godzinie, jeśli konto nie zostanie aktywowane jego rekord jest usuwany z bazy danych, aby nie zupychać niepotrzebnie bazy danych.
5. Użytkownik może natrafić na sytuację, w której nie aktywował konta, a link weryfikacyjny wygasł. Jego rekord jeszcze będzie w bazie, gdyż nie minęła godzina do iteracyjnego usuwania nieuwierzytelnionych użytkowników. Taka sytuacja również została przewidziana. W momencie gdy użytkownik z adresem mailowym nie został aktywowany, a ten email znajduje się już w bazie danych może zarejestrować się ponownie. Jego poprzednia próba i rekord zostanie usunięty, a nowy zapis i token zostanie stworzony.

Rys. 28. Ekran rejestracji

The screenshot shows a registration form titled "REGISTER" in bold capital letters at the top center. Below the title are seven input fields arranged vertically, each preceded by a label: "First Name:", "Last Name:", "Username:", "Email Address:", "Password:", and "Confirm Password:". The "First Name" and "Last Name" fields have placeholder text "First Name" and "Last Name" respectively. The "Username" field has placeholder text "Username". The "Email Address" field has placeholder text "Email Address". The "Password" and "Confirm Password" fields both have placeholder text "Password". At the bottom of the form is a blue rectangular button labeled "Register".

Źródło: Opracowanie własne

Proces logowania

1. Wpisanie przez użytkownika danych logowania: Nazwa użytkownika i hasło
2. Sprawdzenie, czy użytkownik istnieje w systemie. Następnie odhashowywane jest hasło użytkownika o podanym loginie i sprawdzenie, czy zgadza się z tym, co zostało wprowadzone do formularzu.
3. Gdy użytkownik został uwierzytelniony pomyślnie tworzona jest sesja w ciasteczkach dla tego użytkownika przy pomocy JWT.
4. Użytkownik zostaje przeniesiony na stronę główną, gdzie według wygenerowanego tokenu wyświetlane są wszystkie informacje, playlisty i funkcje (zależne od roli użytkownika).

Rys. 29. Ekran logowania

The screenshot shows a login form titled "LOGIN" in bold capital letters at the top center. Below the title are two input fields: "Username:" and "Password:". The "Username" field has placeholder text "Login" and the "Password" field has placeholder text "Password". At the bottom of the form is a blue rectangular button labeled "Login".

Źródło: Opracowanie własne

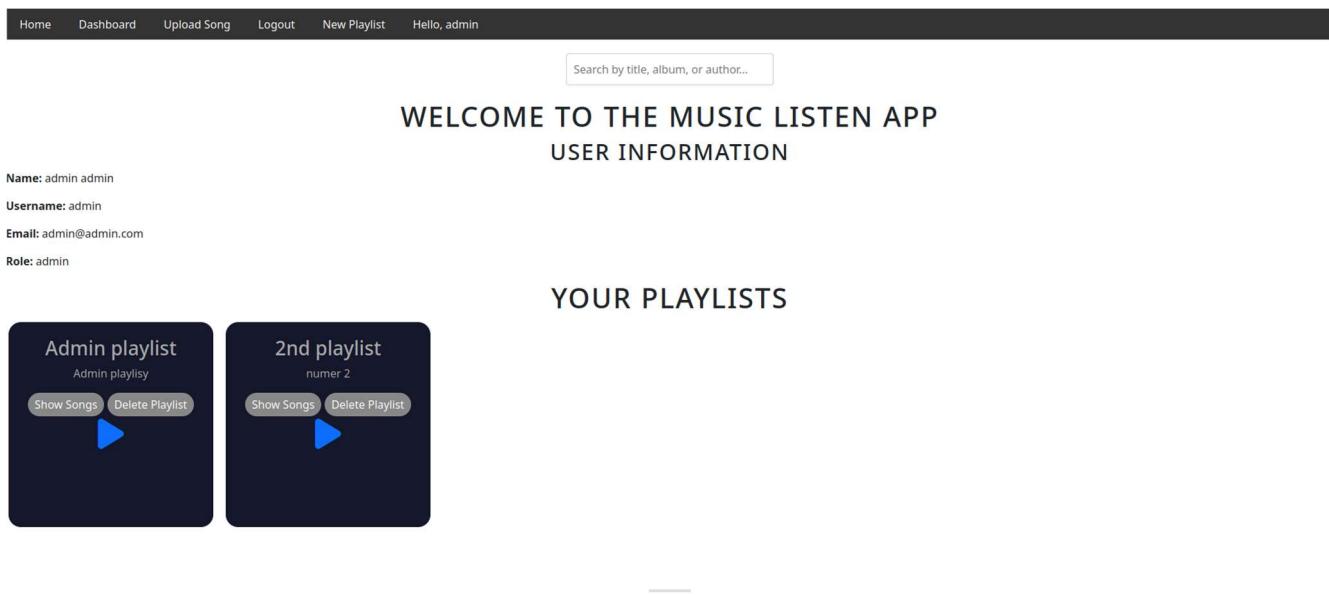
5.1.2 Strona główna aplikacji

Po zalogowaniu użytkownik zostaje przeniesiony na stronę główną aplikacji. Z tego ekranu użytkownik jest w stanie wykonać każdą dla niego akcję. Zaczynając od samej góry, mamy dostępne takie zakładki jak:

- Home – przenosie do strony głównej
- Upload song – otwiera formularz dodania nowej piosenki do serwisu
- Logout – wylogowywuje użytkownika z sesji

Następnie po lewej stronie wyświetlane są informacje użytkownika, takie jak imię i nazwisko, nazwa użytkownika, adres email i rola. Poniżej przycisk otwierający formularz do stworzenia nowej playlisty i sekcja ‘Your playlisys’, w której znajdują się wszystkie składanki użytkownika, wraz z możliwościami usunięcia playlisty, odsłuchania i podglądu utworów w nich zawartych. W górnej części po prawej stronie znajdują się ‘żywa’ wyszukiwarka. To pole oferuje wyszukiwanie w bazie, po wpisaniu tylko frazy piosenki, albumu lub autora. Następnie dynamicznie wyświetlane są najlepsze rezultaty.

Rys. 30. Strona główna

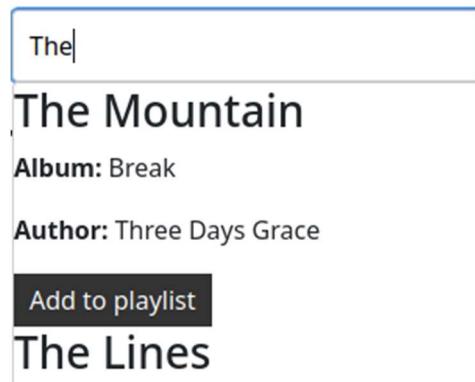


Žródło: Opracowanie własne

5.1.3 Dynamiczne wyszukiwanie utworów

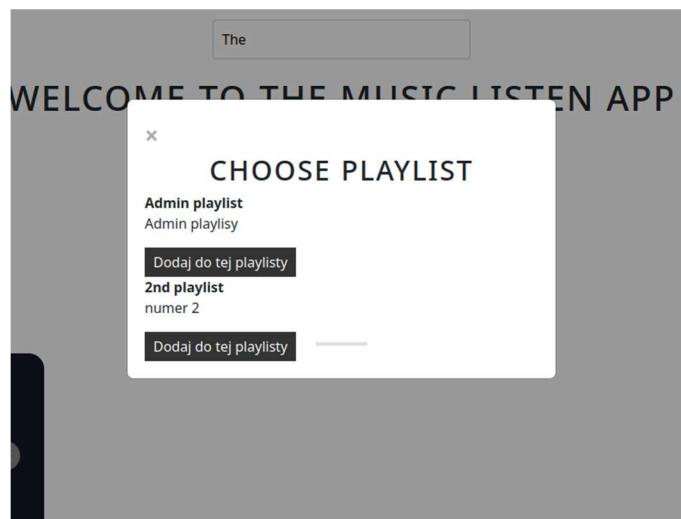
Jak zostało wcześniej wspomniane aplikacja posiada live search. Na poniższym ekranie można zauważyc jedynie frazę znajdująca się w środku tytułu. Umożliwia to użytkownikom znajdowanie piosenek w o wiele łatwiejszy sposób. Wystarczy pewna fraza, a cała piosenka ujawnia się i jest gotowa do dodania do playlisty za pomocą przycisku ‘Add to playlist’. Po kliknięciu przycisku otwiera się okno modal, które ma do wyboru wszystkie playlisty użytkownika.

Rys. 31. Live search



Źródło: Opracowanie własne

Rys. 32. Modal – wybranie do playlisty

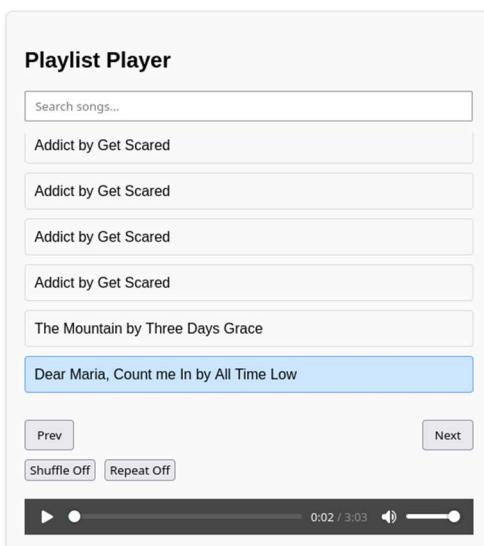


Źródło: Opracowanie własne

5.1.4 Odtwarzacz

Ekran odtwarzacza jest sercem całego projektu. Widok ten nie jest skomplikowany, jest prosty i czytelny dla każdego. Jest on umiejscowiony na samym środku strony, aby skalowanie okna nie zaprzepaściło widoczności. Na samej górze odtwarzacza znajduje się wyszukiwarka. Jest ona prosta i działa jedynie po stronie frontendu, ponieważ obsługuje jedynie rekordy playlisty, które zostały umieszczone w tablicy. W niej wcześniej zostały dodane za pomocą łącznika, tak aby każdy z utworów posiadał swój tytuł i wykonawcę. Utwór właśnie odtwarzany jest podświetlany na niebiesko. W momencie, gdy użytkownik stwierdzi, że chce odsłuchać jednej piosenki w pełni lub zmienić kolejność poprzez odtwarzanie losowe, może wyłączyć te opcje klikając w Repeat lub Shuffle. Odtwarzacz posiada również bazowe funkcje każdego odtwarzacza, takie jak kolejna lub poprzednia piosenka, pauza i start, przewijanie zarówno, jak i kontrolę głośności.

Rys. 33. Odtwarzacz playlist



Źródło: Opracowanie własne

5.1.5 Dodawanie nowego utworu

Niezbędną funkcjonalnością serwisu jest dodawanie nowych utworów do serwisu. Przygotowany został prosty formularz do którego użytkownik wprowadzający dane musi wpisać tytuł, tytuł albumu, wykonawcę, gatunek muzyczny, datę wydania i oczywiście plik. Po kliknięciu w czerwony przycisk sprawdzany jest formularz, czy dane zostały poprawnie wprowadzone, a następnie wysyłany jest nowy wniosek o dodanie utworu do administracji platformy.

Rys. 34. Dodawanie nowego utworu

REQUEST A NEW SONG

The form consists of several input fields:

- Title: [Input field]
- Album: [Input field]
- Author: [Input field]
- Genre: [Dropdown menu showing '--Select a Genre--']
- Release Date: [Input field showing 'dd . mm . rrrr']
- Upload File: [Input field with button 'Przeglądaj...' and placeholder 'Nie wybrano pliku.']

At the bottom is a blue 'Submit Request' button.

Źródło: Opracowanie własne

5.2 Widoki Administratora

Oprócz standardowych widoków, jakie zostały zaprezentowane dla użytkownika systemu administrator posiada dodatkowe ekranie pomagające w administrowaniu aplikacją ze strony frontendu.

5.2.1 Zarządzanie użytkownikami

Ekran ‘Admin Dashboard’ posiada najważniejsze informacje o użytkownikach. Administrator ma do niego dostęp poprzez kliknięcie w zakładkę dashboard, która jest dostępna tylko i wyłącznie dla kont użytkownika z rolą ‘admin’. Jest to działanie odszyfrowujące token użytkownika i jego role zarazem. Baza użytkowników jest obecnie uboga, przez co wszyscy użytkownicy widoczni są na stronie. W momencie, gdy tych użytkowników byłoby więcej zostało przygotowane sortowanie. Administrator może sortować po ID (domyślnie), Imieniu, nazwisku, emailu rosnącą i malejąco. Dostępne jest również wyszukiwanie. Tym razem zostało ono zaprojektowane w sposób, w który to administrator najpierw wybiera, po czym dokładnie chce sortować, a następnie wpisuje frazę i kliką w przycisk Apply, aby zatwierdzić zmiany. W kolumnie ‘Action’ widoczny jest przycisk ‘Show profile’ pozwalający na otworzenie dokładnego profilu wybranego użytkownika, w którym to znajdują się wszystkie dane z bazy danych, a także możliwość usunięcia konta użytkownika. Daje to pełny zarys administrowania kontami użytkowników. Na poniższym ekranie również jest widoczna zakładka ‘Song requests 3’, która to przenosi do panelu administrowania wnioskami o dodanie nowej piosenki do serwisu. Liczba obok ukazuje ilość wniosków ze statusem ‘Pending’, aby administrator był świadomy, że wszystkie wnioski nie zostały jeszcze zatwierdzone lub odrzucone.

Rys. 35. Panel kontrolny administratora - zarządzanie użytkownikami

ID	First Name	Last Name	Username	Email	Verified	Action
5	admin	admin	admin	admin@admin.com	Yes	Show Profile
50	Hubert	Dabek	hdabek	hdabek69@gmail.com	Yes	Show Profile
51	Jacek	Soplica	jacek12	jacek.soplica@email.com	Yes	Show Profile
52	Eliza	Kowalska	eliza23	elizakowalska@exmaple.com	Yes	Show Profile
53	Franciszek	Rogacki	franek333	franek222@exmaple.com	Yes	Show Profile
54	Amelia	Krzysztofik	amelia333	amelicia@exmaple.com	Yes	Show Profile

Źródło: Opracowanie własne

5.2.2 Zarządzanie wnioskami o dodanie nowych utworów

Przechodząc już do ostatniego najważniejszego ekranu aplikacji. Zarządzanie wnioskami użytkowników. Zaraz po otwarciu ekranu zarządzania, administrator jest widokiem tylko i wyłącznie widokiem wszystkich wniosków umieszczonych na samym dole ekranu. Wypisane tam są wszystkie istotne informacje o piosence z wniosku, a także nazwa użytkownika, osoby która zawnioskowała o dodanie utworu.

Proces dodawania nowego utworu.

1. Po kliknięciu w przycisk ‘Play’ otwiera się odtwarzacz, pozwalając na odsłuchanie wybranego dzieła.
2. Jeżeli dane się nie zgadzają administrator, może kliknąć w przycisk ‘Edit’ aby edytować dane.
3. Zaraz po jego kliknięciu otwiera się dynamiczny formularz ‘Edit request’ wypełniony informacjami wniosku, obok którego został przyciśnięty przycisk ‘Edit’.
4. Po zakończonej edycji administrator zapisuje zmiany za pomocą ‘Save changes’. Ten klawisz wywołuje funkcje sprawdzenia autora i albumu, czy takie rekordy już istnieją w bazie dla ułatwienia pracy administratora.
5. Jeżeli funkcja nie znalazła odpowiednich rezultatów, otwiera się kolejny dynamiczny formularz o nazwie ‘Add author’ i ‘Add album’ (warto zaznaczyć, że otwierają się w zależności, którego akurat brakuje). Owa metoda porównuje rekordy po nazwie po stronie backendu.
6. Po pomyślnym dodaniu albumu i wykonawcy ostatnią akcją, dla administratora jest do ‘Add song’, która przenosi wszystkie informacje z tabeli requests do tabeli songs. Jest ona udostępniona w głównym ekranie aplikacji. W ekranie ‘Song requests’ istnieją takie same zasady wyszukiwania jak w Zarządzaniu użytkownikami.

Rys. 36. Panel kontrolny administratora - zarządzanie wnioskami

The screenshot shows a web-based application for managing song requests. At the top, there's a navigation bar with links: Home, Dashboard, Add New Song, Song requests 3, Hello, admin, and Logout. Below the navigation is a title 'SONG REQUESTS REQUEST MANAGEMENT'. A search bar allows users to search by term, title, ID, or ascending/descending order, with an 'Apply' button. There are two sections for adding new entries: 'Add Author' (with a dropdown for 'Author Name' set to 'Nirvana') and 'Add Album' (with dropdowns for 'Album Name' ('Nobody stop us'), 'Release Date' ('dd . mm . yyyy'), and 'Genre' ('Classical music')). A 'Submit' button is present. Below these sections is a 'EDIT REQUEST' form with fields for Title ('The Mountain'), Album ('Nobody stop us'), Author ('Nirvana'), Genre ('Country'), Release Date ('29 . 01 . 2025'), Status ('Approved'), and buttons for Save Changes and Cancel. A large table lists 11 songs with columns for ID, Title, Album, Author, Genre, Release Date, Status, Requester, Song, and Action. The table rows are numbered 7 through 11. Each row contains a 'Play' button, an 'Edit' button, and an 'Add Song' button. The last row (ID 11) has a song title 'Dear Maria, Count me In' and an author 'Dear Maria, Count me In'. The bottom of the page features a media-style control bar with a play/pause button, volume icon, and progress bar.

ID	Title	Album	Author	Genre	Release Date	Status	Requester	Song	Action
7	Addict	Get Scared	Get Scared	Punk Rock	2023-12-23T00:00:00.000Z	Pending	admin	<input type="button" value="Play"/>	<input type="button" value="Edit"/> <input type="button" value="Add Song"/>
8	The Mountain	Break	Three Days Grace	Country	2022-11-21T00:00:00.000Z	Approved	admin	<input type="button" value="Play"/>	<input type="button" value="Edit"/> <input type="button" value="Add Song"/>
9	The Lines	The Lines Mix	Beartooth	Punk Rock	2023-10-20T00:00:00.000Z	Pending	admin	<input type="button" value="Play"/>	<input type="button" value="Edit"/> <input type="button" value="Add Song"/>
10	World So Cold	12 Stones	12 Stones	Rock	2023-11-22T00:00:00.000Z	Pending	admin	<input type="button" value="Play"/>	<input type="button" value="Edit"/> <input type="button" value="Add Song"/>
11	Dear Maria, Count me In	Dear Maria, Count me In	All Time Low	Latin music	2024-12-23T00:00:00.000Z	Approved	admin	<input type="button" value="Play"/>	<input type="button" value="Edit"/> <input type="button" value="Add Song"/>

źródło: Opracowanie własne

6. Zakończenie

Podsumowując, aplikacja stworzona przy użyciu technologii HTML, CSS i JavaScript jako warstwy frontendu oraz Node.js z Express jako backendu, jest nowoczesnym rozwiązaniem spełniającym potrzeby użytkowników w zakresie wygodnego odtwarzania muzyki. System wykorzystuje PostgreSQL, jako bazę danych, co zapewnia stabilność, wydajność i możliwość zarządzania dużymi zbiorami muzyki oraz danymi użytkowników. Do przechowywania plików audio posłużył tutaj AWS S3. Był on świetnym i nowoczesnym wyborem jako serwis hostowania plików. Dzięki zastosowaniu Dockera, aplikacja jest łatwa w wdrożeniu i może być uruchamiana w dowolnym środowisku, co zwiększa jej przenośność i niezależność od specyfikacji serwera.

Dzięki zastosowaniu Node.js i Express, aplikacja charakteryzuje się szybkim czasem odpowiedzi oraz efektywnym zarządzaniem zapytaniami do serwera. Z kolei interfejs użytkownika stworzony w HTML, CSS i JavaScript jest intuicyjny i przyjazny, co wpływa na pozytywne doświadczenia użytkowników. Stylowy design w połączeniu z dynamicznymi funkcjami frontendu zapewnia płynne przechodzenie między utworami oraz dodatkowe opcje, takie jak wyszukiwanie czy tworzenie playlist.

Aplikacja została zaprojektowana z myślą o responsywności, co sprawia, że jest dostępna zarówno na komputerach, jak i urządzeniach mobilnych. Zintegrowane rozwiązania backendowe i frontendowe umożliwiają użytkownikom personalizację ustawień oraz zarządzanie własnymi bibliotekami muzycznymi. Wykorzystanie PostgreSQL zapewnia bezpieczne przechowywanie danych oraz możliwość ich skalowalności w miarę rozwoju aplikacji.

Docker ułatwia z kolei wdrożenie i utrzymanie aplikacji w środowiskach produkcyjnych, minimalizując ryzyko błędów wynikających z różnic w konfiguracji systemów. Dzięki zastosowaniu kontenerów, cały proces uruchamiania aplikacji staje się szybszy i bardziej niezawodny. Rozwiązania te czynią aplikację nie tylko funkcjonalną, ale także skalowalną i gotową na przyszłe rozszerzenia.

Warto także podkreślić, że aplikacja jest elastyczna i może być rozwijana o kolejne funkcje, takie jak rekommendacje muzyczne, tryb offline czy integracja z zewnętrznymi serwisami streamingowymi. Zastosowane technologie pozwalają na łatwe aktualizacje i rozwój aplikacji w odpowiedzi na potrzeby użytkowników. Dodatkowo, przemyślana architektura aplikacji sprawia, że kod jest przezroczysty i łatwy w utrzymaniu przez programistów.

Aplikacja ta stanowi doskonały przykład wykorzystania współczesnych technologii webowych do stworzenia narzędzia o wysokiej wartości użytkowej. Dzięki odpowiedniemu doborowi technologii oraz solidnemu zapleczu technicznemu aplikacja wyróżnia się na tle konkurencji i spełnia standardy nowoczesnych rozwiązań cyfrowych.

Spis ilustracji

Rys. 1. Diagram przypadków użycia użytkowników	15
Rys. 2. Diagram sekwencji logowania użytkownika niezalogowanego.....	16
Rys. 3. Diagram sekwencji przeglądanie katalogu muzyki.....	17
Rys. 4. Diagram sekwencji odtwarzania utworu	18
Rys. 5. Diagram klas	19
Rys. 6. Schemat encji bazy danych	23
Rys. 7. Backend – Endpoint register	25
Rys. 8. Backend – Endpoint streamingu utworu	27
Rys. 9. Backend - Funkcja do weryfikacji czy użytkownik jest administratorem.....	28
Rys. 10. Backend - Połączenie pomiędzy serwerem a serwisem AWS.	29
Rys. 11. Backend - Konfiguracja nodemailer'a.....	29
Rys. 12. Frontend - Strona główna. Sprawdzanie czy użytkownik jest zalogowany	31
Rys. 13. Frontend - Funkcja odpowiedzialna za Live-Search	32
Rys. 14. Frontend - Tworzenie nowej playlisty.....	32
Rys. 15. Frontend - Wyświetlanie wszystkich playlist należących do zalogowanego użytkownika.....	33
Rys. 16. Frontend – Odtwarzacz muzyki	34
Rys. 17. Frontend – funkcje odpowiedzialne za odtwarzanie losowe i odtwarzanie w pętli.	35
Rys. 18. Frontend – Wyszukiwanie utworów w playliście	35
Rys. 19. Frontend – Panel administratora – wypisanie wszystkich użytkowników	36
Rys. 20. Frontend – panel administratora – profil użytkownika	37
Rys. 21. Frontend – sprawdzenie istniejących danych	38
Rys. 22. Frontend – funkcja otwierająca panel dodawania nowego albumu i wykonawcy	38
Rys. 23. Frontend – odwołanie się do backendu w celu sprawdzenia i dodania danych	39
Rys. 24. Test jednostkowy endpointu current-user	40
Rys. 25. Test jednostkowy endpointu login	40
Rys. 26. Test jednostkowy endpointu add-album.....	41
Rys. 27. Test jednostkowy bazy danych.....	41
Rys. 28. Ekran rejestracji.....	43
Rys. 29. Ekran logowania.....	43
Rys. 30. Strona główna.....	44
Rys. 31. Live search	45
Rys. 32. Modal – wybranie do playlisty	45
Rys. 33. Odtwarzacz playlist	46
Rys. 34. Dodawanie nowego utworu.....	47
Rys. 35. Panel kontrolny administratora - zarządzanie użytkownikami.....	48
Rys. 36. Panel kontrolny administratora - zarządzanie wnioskami.....	49

Literatura

1. Oracle, *What is JSON?*, dostępne online: <https://www.oracle.com/pl/database/what-is-json/>, dostęp: 24.05.2024.
2. Spotify Support, *What is Spotify?*, dostępne online: <https://support.spotify.com/pl/article/what-is-spotify/>, dostęp: 26.05.2024.
3. Google Support, *YouTube Music Help*, dostępne online: <https://support.google.com/youtubemusic/answer/6312991?hl=pl>, dostęp: 28.05.2024.
4. eMastered, *Royalty-free music on SoundCloud*, dostępne online: <https://emastered.com/pl/blog/royalty-free-music-soundcloud>, dostęp: 09.02.2025.
5. UDI Group, *Język HTML – co to jest, do czego służy, jak wygląda?*, dostępne online: <https://udigroup.pl/blog/jazyk-html-co-to-jest-do-czego-sluzyc-jak-wyglada/#co-to-jest-html>, dostęp: 15.06.2024.
6. Kompan.pl, *Co to jest CSS?*, dostępne online: <https://kompan.pl/co-to-jest/css/>, dostęp: 15.06.2024.
7. Bootstrap, *Getting Started*, dostępne online: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>, dostęp: 15.06.2024.
8. Mozilla, *JavaScript Guide*, dostępne online: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction?retiredLocale=pl>, dostęp: 15.06.2024.
9. Semcore, *Backend – co to jest i o czym wiedzieć?*, dostępne online: <https://semcore.pl/backend-co-to-jest-i-o-czym-wiedziec/>, dostęp: 15.06.2024.
10. PostgreSQL, *Oficjalna strona PostgreSQL*, dostępne online: <https://www.postgresql.org.pl/>, dostęp: 27.01.2025.
11. Amazon S3, *What is Amazon S3?*, dostępne online: <https://aws.amazon.com/s3/>, dostęp: 22.02.2025.
12. Node.js, *Introduction to Node.js*, dostępne online: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>, dostęp: 26.01.2025.
13. Express.js, *Routing Guide*, dostępne online: <https://expressjs.com/en/guide/routing.html>, dostęp: 22.02.2025.
14. Visual Studio Code, *Documentation*, dostępne online: <https://code.visualstudio.com/docs>, dostęp: 22.02.2025.
15. Git, *Branching and Merging*, dostępne online: <https://git-scm.com/about/branching-and-merging>, dostęp: 22.02.2025.
16. Docker, *Docker Overview*, dostępne online: <https://docs.docker.com/get-started/docker-overview/>, dostęp: 22.02.2025.
17. MVC Pattern, *Wzorzec projektowy MVC*, dostępne online: <https://www.geeksforgeeks.org/mvc-design-pattern/>, dostęp: 22.02.2025.
18. Singleton Pattern, *Singleton*, dostępne online: <https://refactoring.guru/design-patterns/singleton>, dostęp: 22.02.2025.

Streszczenie pracy dyplomowej

Internetowy serwis muzyczny

Autor: Hubert Dąbek

Promotor: Dr inż. Leszek Gajecki

Słowa kluczowe: *Internetowy serwis muzyczny, Przeglądarka, Frontend, Backend, JavaScript, HTML, Node.js*

Aplikacja powstała z użycia rozwiązań frontendowych i backendowych. Przechowuje informacje o użytkowniku wraz z playlistami, które osoba może sama tworzyć. Strona oferuje autoryzacyjną rejestrację i logowanie, co daje poczucie większego bezpieczeństwa danych osobowych. Przez dwie role użytkowników aplikacja jest samo wystarczalna i nie potrzebuje otwartego kodu do zarządzania danymi. Streamowanie muzyki jest możliwe dzięki integralności serwisu z hostingiem plików jakim jest AWS S3.